Introduction

Business Problem

COVID-19 vaccines became available to subsets of the American public at the end of 2020. States and cities differed in terms of which groups of people were initially eligible, and the timeline for expanding eligibility, but by April 19th, all adults were eligible.

Appointment availability was initially scarce in some places, but this very quickly shifted to a growing number of appointments going unfilled across the country. Only 61% of adults have received at least one dose (as of this writing, on May 22th, 2021), so there are still plenty of people who haven't received any doses, even though 82% of people in the latest survey say they definitely or probably intend to.

Vaccination rate is an important metric being considered as state and local governments decide how and when to relax mask-wearing, occupancy, and other COVID restrictions. These entities are already wondering about the best methods to encourage their populations to get vaccinated, and starting campaigns offering perks such as cash and food vouchers. It's worth asking whether these are likely to be the most effective strategies, or whether there are other approaches to consider.

To aid in this understanding, I will build a predictive model to classify whether someone is likely to be vaccine optimistic, or vaccine hesitant, based on their other answers to the Household Pulse Survey that the US Census has been conducting bi-weekly during the pandemic. Interpreting the factors which are most important in predicting vaccine hesitancy among populations may provide insights that will allow state and local governments to develop effective outreach strategies.

OBTAIN

Data Understanding

For this analysis, I will use a publicly available microdata sample from the bi-weekly Household Pulse Surveys that the US Census Bureau has been conducting during the pandemic. Each observation represents an individual's anonymized response to the survey.

Respondents were randomly sampled from all 50 states and the top 15 metropolitan areas, and asked to fill out the survey online. They were asked a variety of questions about their household, including whether they've received any doses of the vaccine, and if not, what their intent was to do so in the future. I will engineer these features into a binary target variable representing vaccine optimism versus vaccine hesitancy, and use each respondent's answers to other survey questions as the predictors.

Some survey questions were only presented to the respondent if they answered a certain way to a prior question, such as only asking about intention to get a vaccine in the future if the respondent indicated they had not yet received any doses. I will carefully review the data dictionary provided by the Census Bureau to understand these relationships, and engineer features from the groups of questions as appropriate.

I will use the most recent microdata file available as of this writing, May 14, 2021, which includes a sample of about 78,000 responses from Phase 3 of the survey, covering the timeframe of March 3 to March 15, 2021. This is referred to as the "week 26" period. Note that the survey is ongoing and currently in Phase 3.1, but microdata was not yet available for the "week 27" period or first period of Phase 3.1, so I have used the most recent data available at the time.

For more technical information on the sampling methodology and full questionnaire, see here.

In [968…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import ticker
%matplotlib inline

import seaborn as sns
import missingno
import plotly.express as px
import dstools
from textwrap import wrap

pd.set_option('display.max_info_rows', 300)
pd.set_option('display.max_info_columns', 300)

%autoreload 2
```

In [272…

```
# import data from downloaded CSV
df = pd.read_csv(r"data/pulse2021_puf_26.csv.tar.gz")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 78307 entries, 0 to 78306
Data columns (total 204 columns):
 #   Column                  Dtype  
---  ------                  -----  
 0   ._pulse2021_puf_26.csv  object 
 1   WEEK                    float64
 2   EST_ST                  float64
 3   EST_MSA                 float64
 4   REGION                  float64
 5   HWEIGHT                 float64
 6   PWEIGHT                 float64
 7   TBIRTH_YEAR             float64
 8   ABIRTH_YEAR             float64
 9   EGENDER                 float64
 10  AGENDER                 float64
 11  RHISPANIC               float64
 12  AHISPANIC               float64
 13  RRACE                   float64
 14  ARACE                   float64
 15  EEDUC                   float64
 16  AEDUC                   float64
 17  MS                      float64
```

18	THHLD_NUMPER	float64
19	AHHLD_NUMPER	float64
20	THHLD_NUMKID	float64
21	AHHLD_NUMKID	float64
22	THHLD_NUMADLT	float64
23	RECVDVACC	float64
24	DOSES	float64
25	GETVACC	float64
26	WHYNOT1	float64
27	WHYNOT2	float64
28	WHYNOT3	float64
29	WHYNOT4	float64
30	WHYNOT5	float64
31	WHYNOT6	float64
32	WHYNOT7	float64
33	WHYNOT8	float64
34	WHYNOT9	float64
35	WHYNOT10	float64
36	WHYNOT11	float64
37	WHYNOTB1	float64
38	WHYNOTB2	float64
39	WHYNOTB3	float64
40	WHYNOTB4	float64
41	WHYNOTB5	float64
42	WHYNOTB6	float64
43	HADCOVID	float64
44	WRKLOSS	float64
45	EXPCTLOSS	float64
46	ANYWORK	float64
47	KINDWORK	float64
48	RSNNOWRK	float64
49	TW_START	float64
50	UI_APPLY	float64
51	UI_RECV	float64
52	SSA_RECV	float64
53	SSA_APPLY	float64
54	SSAPGM1	float64
55	SSAPGM2	float64
56	SSAPGM3	float64
57	SSAPGM4	float64
58	SSAPGM5	float64
59	SSALIKELY	float64
60	SSAEXPCT1	float64
61	SSAEXPCT2	float64
62	SSAEXPCT3	float64
63	SSAEXPCT4	float64
64	SSAEXPCT5	float64
65	SSADECISN	float64
66	EIP	float64
67	EIPSPND1	float64
68	EIPSPND2	float64
69	EIPSPND3	float64
70	EIPSPND4	float64
71	EIPSPND5	float64
72	EIPSPND6	float64
73	EIPSPND7	float64
74	EIPSPND8	float64
75	EIPSPND9	float64
76	EIPSPND10	float64
77	EIPSPND11	float64
78	EIPSPND12	float64
79	EIPSPND13	float64
80	EXPNS_DIF	float64
81	CHNGHOW1	float64
82	CHNGHOW2	float64

```
83  CHNGHOW3              float64
84  CHNGHOW4              float64
85  CHNGHOW5              float64
86  CHNGHOW6              float64
87  CHNGHOW7              float64
88  CHNGHOW8              float64
89  CHNGHOW9              float64
90  CHNGHOW10             float64
91  CHNGHOW11             float64
92  CHNGHOW12             float64
93  WHYCHNGD1             float64
94  WHYCHNGD2             float64
95  WHYCHNGD3             float64
96  WHYCHNGD4             float64
97  WHYCHNGD5             float64
98  WHYCHNGD6             float64
99  WHYCHNGD7             float64
100 WHYCHNGD8             float64
101 WHYCHNGD9             float64
102 WHYCHNGD10            float64
103 WHYCHNGD11            float64
104 WHYCHNGD12            float64
105 WHYCHNGD13            float64
106 SPNDSRC1              float64
107 SPNDSRC2              float64
108 SPNDSRC3              float64
109 SPNDSRC4              float64
110 SPNDSRC5              float64
111 SPNDSRC6              float64
112 SPNDSRC7              float64
113 SPNDSRC8              float64
114 FEWRTRIPS             float64
115 FEWRTRANS             float64
116 PLNDTRIPS             float64
117 CURFOODSUF            float64
118 CHILDFOOD             float64
119 FOODSUFRSN1           float64
120 FOODSUFRSN2           float64
121 FOODSUFRSN3           float64
122 FOODSUFRSN4           float64
123 FOODSUFRSN5           float64
124 FREEFOOD              float64
125 WHEREFREE1            float64
126 WHEREFREE2            float64
127 WHEREFREE3            float64
128 WHEREFREE4            float64
129 WHEREFREE5            float64
130 WHEREFREE6            float64
131 WHEREFREE7            float64
132 SNAP_YN               float64
133 TSPNDFOOD             float64
134 TSPNDPRPD             float64
135 ANXIOUS               float64
136 WORRY                 float64
137 INTEREST              float64
138 DOWN                  float64
139 HLTHINS1              float64
140 HLTHINS2              float64
141 HLTHINS3              float64
142 HLTHINS4              float64
143 HLTHINS5              float64
144 HLTHINS6              float64
145 HLTHINS7              float64
146 HLTHINS8              float64
147 PRIVHLTH              float64
```

```
 148 PUBHLTH                 float64
 149 DELAY                   float64
 150 NOTGET                  float64
 151 PRESCRIPT               float64
 152 MH_SVCS                 float64
 153 MH_NOTGET               float64
 154 TENURE                  float64
 155 LIVQTR                  float64
 156 RENTCUR                 float64
 157 MORTCUR                 float64
 158 MORTCONF                float64
 159 EVICT                   float64
 160 FORCLOSE                float64
 161 ENROLL1                 float64
 162 ENROLL2                 float64
 163 ENROLL3                 float64
 164 TEACH1                  float64
 165 TEACH2                  float64
 166 TEACH3                  float64
 167 TEACH4                  float64
 168 TEACH5                  float64
 169 COMPAVAIL               float64
 170 COMP1                   float64
 171 COMP2                   float64
 172 COMP3                   float64
 173 INTRNTAVAIL             float64
 174 INTRNT1                 float64
 175 INTRNT2                 float64
 176 INTRNT3                 float64
 177 SCHLHRS                 float64
 178 TSTDY_HRS               float64
 179 TCH_HRS                 float64
 180 TNUM_PS                 float64
 181 PSPLANS1                float64
 182 PSPLANS2                float64
 183 PSPLANS3                float64
 184 PSPLANS4                float64
 185 PSPLANS5                float64
 186 PSPLANS6                float64
 187 PSCHNG1                 float64
 188 PSCHNG2                 float64
 189 PSCHNG3                 float64
 190 PSCHNG4                 float64
 191 PSCHNG5                 float64
 192 PSCHNG6                 float64
 193 PSCHNG7                 float64
 194 PSWHYCHG1               float64
 195 PSWHYCHG2               float64
 196 PSWHYCHG3               float64
 197 PSWHYCHG4               float64
 198 PSWHYCHG5               float64
 199 PSWHYCHG6               float64
 200 PSWHYCHG7               float64
 201 PSWHYCHG8               float64
 202 PSWHYCHG9               float64
 203 INCOME                  float64
dtypes: float64(203), object(1)
memory usage: 121.9+ MB
```

In [273…

```
# drop placeholder column
df.drop(columns=['._pulse2021_puf_26.csv'], inplace=True)
df
```

Out[273…

	WEEK	EST_ST	EST_MSA	REGION	HWEIGHT	PWEIGHT	TBIRTH_YEAR	ABIRTH_YI

	WEEK	EST_ST	EST_MSA	REGION	HWEIGHT	PWEIGHT	TBIRTH_YEAR	ABIRTH_YI
0	26.0	1.0	NaN	2.0	1546.046792	2976.976997	1956.0	
1	26.0	1.0	NaN	2.0	3481.483789	6703.740928	1995.0	
2	26.0	1.0	NaN	2.0	581.457250	1119.619967	1961.0	
3	26.0	1.0	NaN	2.0	2462.927760	7113.703795	1971.0	
4	26.0	1.0	NaN	2.0	629.103937	605.682816	1941.0	
...	...	...	...	...	...	...	...	
78302	26.0	56.0	NaN	4.0	182.025916	168.795754	1985.0	
78303	26.0	6.0	NaN	4.0	671.196803	1281.472182	1997.0	
78304	26.0	26.0	19820.0	3.0	320.220440	597.103115	1992.0	
78305	26.0	56.0	NaN	4.0	127.938345	237.278844	1981.0	
78306	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

78307 rows × 203 columns

I will primarily use the data dictionary provided by the Census Bureau to determine which fields to include in my initial model.

Although all the data fields are stored as floats, based on the dictionary some of them are actually ordinally encoded categorical variables, and most are nominal categorical variables. I will need to determine which are which, and whether any can be treated as continuous or numeric variables.

SCRUB

I will need to engineer multiple features, including my target. As written and organized into a question per column, the data is not in ideal format for analysis due to the presence of main question and sub-question groups; some questions are conceptually spread across multiple columns.

Since predictors are mostly categorical, I'm concerned about the size of the feature space when they are eventually one-hot-encoded. As I scrub and explore this data, I'll engineer features that I think might be simpler alternatives to the originals but still capture the main information. I can experiment with the engineered features versus original ones in the modeling phase to let the models decide which, if any, perform better.

One notable challenge will be dealing with missing data. The data dictionary indicates that there are two placeholders that represent a non-answer:

- -88: Missing / Did not report
- -99: Question seen but category not selected

The Census Bureau provides no additional information in the data dictionary about why a response might be missing and placed in the -88 category as opposed to the -99 category that represents a lack of a response from the respondent. Although some instances appear to be filled in to account for the fact that a sub-question wasn't presented to a respondent because of how they answered a prior question, there are some columns with -88 placeholders that do not appear to be sub-questions, so in my understanding should have been presented to everyone.

Since there are a significant number of these -88 placeholders in some top-level question columns, I will likely encode them as a separate category instead of dropping them or attempting to impute them. But I will explore further before deciding how to proceed.

Target columns

I want to create a binary target that represents a respondent's vaccine outlook: hesitant, or optimistic.

I will need to engineer this target from two questions:

- `RECVDVACC` asked the respondent whether they have received any doses of the COVID vaccine
- `GETVACC` is a sub-question asked of the respondent if they said they had not received any doses. It asked about the respondent's intent to get a vaccine in the future. Choices include:

1. Definitely get a vaccine
2. Probably get a vaccine
3. Probably NOT get a vaccine
4. Definitely NOT get a vaccine

At the time of the survey back in early March, only certain subsets of the population were eligible to make a vaccine appointment, based on state-specific criteria such as age, occupation, health risk indicators, and other factors. I want to make sure my class takes into account people who have been vaccinated but doesn't penalize people who have not yet been vaccinated yet, given there is no easy way to know if they were eligible for vaccination when they were surveyed.

I will create the binary target by combining answers to these two questions in the following way:

Vaccine optimistic

- Has received at least one dose OR
- Has not received any doses yet AND:
  - Intends to Definitely get a vaccine OR
  - Intends to Probably get a vaccine

Vaccine hesitant

- Has not received any doses yet AND:

- Intends to Probably NOT get a vaccine OR
- Intends to Definitely NOT get a vaccine

In [274…
```
# First question asking whether they have received any doses
df['RECVDVACC'].value_counts()
```

Out[274…
```
 2.0     42085
 1.0     35840
-99.0      381
Name: RECVDVACC, dtype: int64
```

In [275…
```
df['RECVDVACC'].isna().sum()
```

Out[275… 1

In [276…
```
to_drop = df.loc[df['RECVDVACC'] == -99.0].index
df.drop(index=to_drop, inplace=True)

df['RECVDVACC'].value_counts()
```

Out[276…
```
2.0    42085
1.0    35840
Name: RECVDVACC, dtype: int64
```

- Value of 1 indicates Yes, at least one dose
- Value of 2 indicates No, no doses yet
- Value of -99 indicates the respondent did not answer the question

I will drop the -99 values, since there are few of them and I wouldn't be comfortable imputing target variables.

In [277…
```
df.tail(5)
```

Out[277…

	WEEK	EST_ST	EST_MSA	REGION	HWEIGHT	PWEIGHT	TBIRTH_YEAR	ABIRTH_YEA
78302	26.0	56.0	NaN	4.0	182.025916	168.795754	1985.0	2
78303	26.0	6.0	NaN	4.0	671.196803	1281.472182	1997.0	2
78304	26.0	26.0	19820.0	3.0	320.220440	597.103115	1992.0	2
78305	26.0	56.0	NaN	4.0	127.938345	237.278844	1981.0	2
78306	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

5 rows × 203 columns

It looks like there was an extra blank line at the end of the file that is NaN for all values, so I'll remove that too.

In [278…
```
df.dropna(axis=0, how='all', inplace=True)
```

In [279…
```
df.tail(5)
```

Out[279…

	WEEK	EST_ST	EST_MSA	REGION	HWEIGHT	PWEIGHT	TBIRTH_YEAR	ABIRTH_YEA

	WEEK	EST_ST	EST_MSA	REGION	HWEIGHT	PWEIGHT	TBIRTH_YEAR	ABIRTH_YEA
78301	26.0	56.0	NaN	4.0	122.306386	340.250428	1964.0	2
78302	26.0	56.0	NaN	4.0	182.025916	168.795754	1985.0	2
78303	26.0	6.0	NaN	4.0	671.196803	1281.472182	1997.0	2
78304	26.0	26.0	19820.0	3.0	320.220440	597.103115	1992.0	2
78305	26.0	56.0	NaN	4.0	127.938345	237.278844	1981.0	2

5 rows × 203 columns

In [280…
```
# Review values for second vaccine-related question
df['GETVACC'].value_counts()
```

Out[280…
```
-88.0    35840
 1.0     24576
 2.0      7904
 3.0      5271
 4.0      4234
-99.0      100
Name: GETVACC, dtype: int64
```

In [281…
```
df['GETVACC'].isna().sum()
```

Out[281… 0

This question was only asked of respondents who said they had not received any doses yet.

- Value of 1 indicates will Definitely get a vaccine
- Value of 2 indicates will Probably get a vaccine
- Value of 3 indicates will Probably NOT get a vaccine
- Value of 4 indicates will Definitely NOT get a vaccine
- Value of -99 indicates the respondent did not answer the question

A value of -88 is described as simply missing in the data dictionary. In this case, we know that this question was asked of a subset of people who sad they hadn't received any doses, and the count of value -88 in this question matches the count of 1 - Yes in the previous question.

I will confirm that these -88 placeholders represent people who received a dose of the vaccine.

In [282…
```
# Do 'GETVACC' values of -88 also have 'RECVDVACC' values of 1?
df.loc[df['GETVACC'] == -88.0]['RECVDVACC'].value_counts()
```

Out[282…
```
1.0    35840
Name: RECVDVACC, dtype: int64
```

Yes, all the -88 values from the second question are people who answered Yes (1.0) in the first question.

I will drop the -99 values to the second question since they are also very small in size. Then I will create a binary target based on the logic outlined in the header section above.

In [283…
```
# drop -99 values, where the respondent didn't answer the question
```

```
to_drop = df.loc[df['GETVACC'] == -99.0].index
df.drop(index=to_drop, inplace=True)

# Target class 0 will be vaccine optimistic
df.loc[(df['RECVDVACC'] == 1.0) |
       (df['GETVACC'] == 1.0) |
       (df['GETVACC'] == 2.0), 'target'] = 0

# Target class 1 will be vaccine hesitant
df.loc[(df['GETVACC'] == 3.0) |
       (df['GETVACC'] == 4.0), 'target'] = 1

df['target'].value_counts()
```

Out[283…
```
0.0    68320
1.0     9505
Name: target, dtype: int64
```

In [284…
```
df['target'].value_counts(normalize=True)
```

Out[284…
```
0.0    0.877867
1.0    0.122133
Name: target, dtype: float64
```

In [285…
```
df['target'].isna().sum()
```

Out[285…
```
0
```

Predictor Columns

I used the data dictionary provided by the Census Bureau as the primary method of determining which columns were categorical versus numeric, and which I wanted to use versus leave out of the initial exploration. A copy of the data dictionary is provided in the data folder.

Many of the questions asked in this survey involve potentially sensitive topics, such as race, financial circumstances, participation level in social assistance programs, household spending habits, aspects of mental health, etc. I will therefore do my utmost to apply consistent, objective criteria regarding which columns to include or remove in the model to avoid unintentional personal bias, and will detail my thoughts behind each of these tactical decisions.

Initial Columns to Exclude

As a first pass, I identified the following characteristics of columns in the data set that led me to decide to exclude them in the initial exploration and probably from modeling.

Subquestions

Certain questions were asked of respondents only if they answered in a certain way to a prior question. One example would be the set of `WHYCHNGD` subquestions asking WHY household spending had changed, which were only presented to respondents who said their household spending HAD changed.

I think these subquestions will be very useful in understanding more nuance about the opinions and circumstances of households, if the primary question turns out to be an important predictor of vaccine hesitancy. However, I will include only the top-level questions in the modeling process.

Fields that represent subquestions (based on the data dictionary), and will be excluded from initial analysis are:

- WHYNOT1
- WHYNOT2
- WHYNOT3
- WHYNOT4
- WHYNOT5
- WHYNOT6
- WHYNOT7
- WHYNOT8
- WHYNOT9
- WHYNOT10
- WHYNOT11
- WHYNOTB1
- WHYNOTB2
- WHYNOTB3
- WHYNOTB4
- WHYNOTB5
- WHYNOTB6
- RSNNOWRK
- SSAPGM1
- SSAPGM2
- SSAPGM3
- SSAPGM4
- SSAPGM5
- SSALIKELY
- SSAEXPCT1
- SSAEXPCT2
- SSAEXPCT3
- SSAEXPCT4
- SSAEXPCT5
- SSADECISN
- EIPSPND1
- EIPSPND2
- EIPSPND3
- EIPSPND4
- EIPSPND5
- EIPSPND6
- EIPSPND7

- EIPSPND8
- EIPSPND9
- EIPSPND10
- EIPSPND11
- EIPSPND12
- EIPSPND13
- WHYCHNGD1
- WHYCHNGD2
- WHYCHNGD3
- WHYCHNGD4
- WHYCHNGD5
- WHYCHNGD6
- WHYCHNGD7
- WHYCHNGD8
- WHYCHNGD9
- WHYCHNGD10
- WHYCHNGD11
- WHYCHNGD12
- WHYCHNGD13
- CHILDFOOD
- FOODSUFRSN1
- FOODSUFRSN2
- FOODSUFRSN3
- FOODSUFRSN4
- FOODSUFRSN5
- WHEREFREE1
- WHEREFREE2
- WHEREFREE3
- WHEREFREE4
- WHEREFREE5
- WHEREFREE6
- WHEREFREE7
- RENTCUR
- MORTCUR
- MORTCONF
- EVICT
- FORCLOSE
- TEACH1
- TEACH2
- TEACH3
- TEACH4
- TEACH5
- COMPAVAIL
- COMP1

- COMP2
- COMP3
- INTRNTAVAIL
- INTRNT1
- INTRNT2
- INTRNT3
- KINDWORK
- SCHLHRS
- TSTDY_HRS
- TCH_HRS
- PSPLANS1
- PSPLANS2
- PSPLANS3
- PSPLANS4
- PSPLANS5
- PSPLANS6
- PSCHNG1
- PSCHNG2
- PSCHNG3
- PSCHNG4
- PSCHNG5
- PSCHNG6
- PSCHNG7
- PSWHYCHG1
- PSWHYCHG2
- PSWHYCHG3
- PSWHYCHG4
- PSWHYCHG5
- PSWHYCHG6
- PSWHYCHG7
- PSWHYCHG8
- PSWHYCHG9
- DOSES

Duplicates, Metadata, and Imputed values

Several columns were already imputed into summary columns by the Census Bureau, which are duplicative of the originals:

- THHLD_NUMPER
- PRIVHLTH
- PUBHLTH

The REGION column will also not be included, as the dataset contains other columns that can be used to represent geographical characteristics, such as State and Metropolitan Area.

A few columns represent metadata such as household identifier, and weight calculations to be used to generate population statistics:

- SCRAM
- PWEIGHT
- HWEIGHT
- WEEK

The following columns are indicators that a value was imputed by the Census Bureau:

- ABIRTH_YEAR
- AGENDER
- AHISPANIC
- ARACE
- AEDUC
- AHHLD_NUMPER
- AHHLD_NUMKID

Categories to Engineer

Some categorical columns are likely candidates to be engineered into a single column, such as reducing respondents' answers to different types of health insurance they have into a single column representing whether they have health insurance or not.

I can't be sure yet whether these features will perform better in their original state, or if my engineered features will be useful predictors. I'll create them where they make sense for now, and will be able to model either or both versions.

In [286…

```
# See values in HLTHINS columns
hlth_cols = ['HLTHINS1', 'HLTHINS2', 'HLTHINS3', 'HLTHINS4', 'HLTHINS5',
             'HLTHINS6', 'HLTHINS7', 'HLTHINS8']

for col in hlth_cols:
    print(df[col].value_counts(1))
```

```
 1.0     0.534623
 2.0     0.262987
-88.0    0.169663
-99.0    0.032727
Name: HLTHINS1, dtype: float64
 2.0     0.537861
 1.0     0.194064
-88.0    0.169663
-99.0    0.098413
Name: HLTHINS2, dtype: float64
 2.0     0.494982
 1.0     0.257861
-88.0    0.169663
-99.0    0.077494
Name: HLTHINS3, dtype: float64
 2.0     0.626637
-88.0    0.169663
-99.0    0.118484
 1.0     0.085217
Name: HLTHINS4, dtype: float64
```

```
 2.0     0.667086
-88.0    0.169663
-99.0    0.124664
 1.0     0.038587
Name: HLTHINS5, dtype: float64
 2.0     0.661934
-88.0    0.169663
-99.0    0.129213
 1.0     0.039190
Name: HLTHINS6, dtype: float64
 2.0     0.685744
-88.0    0.169663
-99.0    0.137347
 1.0     0.007247
Name: HLTHINS7, dtype: float64
 2.0     0.628050
-88.0    0.169663
-99.0    0.163996
 1.0     0.038291
Name: HLTHINS8, dtype: float64
```

In [287…

```
# Combine HLTHINS multi-choice answers to represent has insurance or not

# Assign 1 if any health insurance option chosen
df.loc[(df['HLTHINS1']==1)|(df['HLTHINS2']==1)|(df['HLTHINS3']==1)|
       (df['HLTHINS4']==1)|(df['HLTHINS5']==1)|(df['HLTHINS6']==1)|
       (df['HLTHINS7']==1)|(df['HLTHINS8']==1), ['HLTHINS']] = 1

# Assign 0 if No explicitly selected for all
df.loc[(df['HLTHINS1']==2)&(df['HLTHINS2']==2)&(df['HLTHINS3']==2)&
       (df['HLTHINS4']==2)&(df['HLTHINS5']==2)&(df['HLTHINS6']==2)&
       (df['HLTHINS7']==2)&(df['HLTHINS8']==2), ['HLTHINS']] = 0

# Assign -99 if skipped all
df.loc[(df['HLTHINS1']==-99)&(df['HLTHINS2']==-99)&(df['HLTHINS3']==-99)&
       (df['HLTHINS4']==-99)&(df['HLTHINS5']==-99)&(df['HLTHINS6']==-99)&
       (df['HLTHINS7']==-99)&(df['HLTHINS8']==-99), ['HLTHINS']] = -99

# Assign -88 if we have no answer for all
df.loc[(df['HLTHINS1']==-88)&(df['HLTHINS2']==-88)&(df['HLTHINS3']==-88)&
       (df['HLTHINS4']==-88)&(df['HLTHINS5']==-88)&(df['HLTHINS6']==-88)&
       (df['HLTHINS7']==-88)&(df['HLTHINS8']==-88), ['HLTHINS']] = -88

print(df['HLTHINS'].isna().sum())
df['HLTHINS'].value_counts()
```

```
451
```

Out[287…

```
 1.0     61184
-88.0    13204
 0.0      2653
-99.0      333
Name: HLTHINS, dtype: int64
```

In [288…

```
# There are still 451 nulls, let's see how those answers look
df.loc[df['HLTHINS'].isna(), hlth_cols]
```

Out[288…

	HLTHINS1	HLTHINS2	HLTHINS3	HLTHINS4	HLTHINS5	HLTHINS6	HLTHINS7	HLTHINS8
235	2.0	2.0	-99.0	-99.0	-99.0	-99.0	-99.0	-99.0
299	2.0	2.0	2.0	2.0	2.0	2.0	-99.0	-99.0
490	2.0	-99.0	-99.0	-99.0	-99.0	-99.0	-99.0	-99.0

	HLTHINS1	HLTHINS2	HLTHINS3	HLTHINS4	HLTHINS5	HLTHINS6	HLTHINS7	HLTHINS8
558	2.0	2.0	2.0	2.0	2.0	2.0	2.0	-99.0
604	2.0	2.0	2.0	2.0	2.0	2.0	2.0	-99.0
...	...	...	...	...	...	...	...	...
77636	2.0	-99.0	-99.0	-99.0	-99.0	-99.0	-99.0	-99.0
77848	2.0	2.0	-99.0	-99.0	-99.0	-99.0	-99.0	-99.0
77877	2.0	-99.0	-99.0	-99.0	-99.0	-99.0	-99.0	-99.0
77976	-99.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0
78048	2.0	-99.0	-99.0	-99.0	-99.0	-99.0	-99.0	-99.0

451 rows × 8 columns

In [289…
```
# Assign the remaining records to 0, since it looks like respondents
# started marking them all No (2) but skipped some (-99)
df.loc[df['HLTHINS'].isna(), 'HLTHINS'] = 0
```

The questions about children enrolled in school are also a good candidate for an engineered feature. I'll create a single binary feature representing whether the household had any children enrolled in external school (i.e.not homeschooled, since homeschooling was likely less disrupted by the pandemic restrictions).

In [290…
```
# Create a single categorical column for children enrolled in school outside
# the home

# Assign 1 if have any children enrolled in school outside the home
df.loc[(df['ENROLL1']==1), ['SCHOOL_KIDS']] = 1

# Assign 0 if have ONLY children who are being homeschooled
df.loc[(df['ENROLL2']==1) & (df['SCHOOL_KIDS'].isna()), ['SCHOOL_KIDS']] = 0

# Assign 3 if indicated did NOT have children enrolled in school
df.loc[(df['ENROLL3']==1) & (df['SCHOOL_KIDS'].isna()), ['SCHOOL_KIDS']] = 0

print(len(df.loc[df['SCHOOL_KIDS'].isna(), ['SCHOOL_KIDS']]))
df['SCHOOL_KIDS'].value_counts()
```

```
57699
```

Out[290…
```
1.0    14266
0.0     5860
Name: SCHOOL_KIDS, dtype: int64
```

In [291…
```
# wow that's a lot of nulls. What were the answers?
df.loc[df['SCHOOL_KIDS'].isna(), ['ENROLL1', 'ENROLL2', 'ENROLL3']].value_counts
```

Out[291…
```
ENROLL1  ENROLL2  ENROLL3
-88.0    -88.0    -88.0      57437
-99.0    -99.0    -99.0        262
dtype: int64
```

I'm definitely going to need to look more into the -88 values, since there are a huge number of them for these questions.

Is it possible that these questions were only asked of respondents who said they did have kids?

In [292…
```
df.loc[(df['ENROLL1']==-88) & (df['ENROLL2']==-88) & (df['ENROLL3']==-88),
       'THHLD_NUMKID'].value_counts()
```

Out[292…
```
0.0    51941
1.0     2564
2.0     1882
3.0      682
4.0      226
5.0      142
Name: THHLD_NUMKID, dtype: int64
```

Most of the -88 values represent respondents who reported 0 people under 18 in the household so I'll mark those as 0 in my engineered feature for No.

The rest, I'll mark as -88 or -99 as the original answers were and deal with them later.

In [293…
```
# Assign 0 if said they had no children in the household
df.loc[(df['SCHOOL_KIDS'].isna()) & (df['THHLD_NUMKID']==0),
       ['SCHOOL_KIDS']] = 0

# Assign -88 and -99 as appropriate
df.loc[(df['ENROLL1']==-88) & (df['ENROLL2']==-88) & (df['ENROLL3']==-88) &
       (df['SCHOOL_KIDS'].isna()), ['SCHOOL_KIDS']] = -88

df.loc[(df['ENROLL1']==-99) & (df['ENROLL2']==-99) & (df['ENROLL3']==-99) &
       (df['SCHOOL_KIDS'].isna()), ['SCHOOL_KIDS']] = -99

print(len(df.loc[df['SCHOOL_KIDS'].isna(), ['SCHOOL_KIDS']]))
df['SCHOOL_KIDS'].value_counts()
```

```
0
```

Out[293…
```
 0.0     57873
 1.0     14266
-88.0     5496
-99.0      190
Name: SCHOOL_KIDS, dtype: int64
```

In [294…
```
# Combine TSPNDPRPD and TSPNDFOOD into a single column representing the
# proportion of money spent on food that was on food to be eaten at home
# TSPNDPRPD is amount spent on prepared meals
# TSPNDFOOD is amount spent on food to be prepared at home

# if respondent entered one food amount but not the other, assume they left
# the other blank because it should be 0
df.loc[(df['TSPNDPRPD'] < 0) & (df['TSPNDFOOD'] >0), 'TSPNDPRPD'] = 0
df.loc[(df['TSPNDPRPD'] > 0) & (df['TSPNDFOOD'] <0), 'TSPNDFOOD'] = 0

# Only calculate if respondent included an answer for one
ans = df.loc[(df['TSPNDPRPD'] >0) | (df['TSPNDFOOD'] >0), ['TSPNDPRPD', 'TSPNDFO
ans.head()
```

Out[294…

	TSPNDPRPD	TSPNDFOOD
1	50.0	200.0
3	0.0	300.0
4	210.0	100.0

	TSPNDPRPD	TSPNDFOOD
5	200.0	200.0
6	0.0	40.0

In [295…
```
# Populate proportion of food spend that was on food to be prepared
# at home
ans['PROP_FOODSPEND_HOME'] = ans.apply(lambda x: x[1]/np.sum(x), axis=1)
ans.head()
```

Out[295…

	TSPNDPRPD	TSPNDFOOD	PROP_FOODSPEND_HOME
1	50.0	200.0	0.800000
3	0.0	300.0	1.000000
4	210.0	100.0	0.322581
5	200.0	200.0	0.500000
6	0.0	40.0	1.000000

In [296…
```
# merge results back into main df
df = df.merge(right=ans['PROP_FOODSPEND_HOME'], how='left',
         left_on=df.index, right_on=ans.index)

df[['TSPNDPRPD','TSPNDFOOD','PROP_FOODSPEND_HOME']].head(20)
```

Out[296…

	TSPNDPRPD	TSPNDFOOD	PROP_FOODSPEND_HOME
0	-88.0	-88.0	NaN
1	50.0	200.0	0.800000
2	-88.0	-88.0	NaN
3	0.0	300.0	1.000000
4	210.0	100.0	0.322581
5	200.0	200.0	0.500000
6	0.0	40.0	1.000000
7	0.0	800.0	1.000000
8	8.0	84.0	0.913043
9	25.0	120.0	0.827586
10	-99.0	-99.0	NaN
11	50.0	150.0	0.750000
12	20.0	300.0	0.937500
13	0.0	200.0	1.000000
14	0.0	200.0	1.000000
15	50.0	280.0	0.848485
16	0.0	150.0	1.000000
17	0.0	70.0	1.000000

	TSPNDPRPD	TSPNDFOOD	PROP_FOODSPEND_HOME
18	250.0	90.0	0.264706
19	200.0	200.0	0.500000

In [297…
```
# Insert appropriate -99 and -88 values, so can assess -88s in bulk
# with other columns later
df.loc[(df['TSPNDPRPD'] ==-88) & (df['TSPNDFOOD'] ==-88),
       'PROP_FOODSPEND_HOME'] = -88
df.loc[(df['TSPNDPRPD'] ==-99) & (df['TSPNDFOOD'] ==-99),
       'PROP_FOODSPEND_HOME'] = -99
```

In [298…
```
# review remaining null values
df.loc[df['PROP_FOODSPEND_HOME'].isna(),
       ['TSPNDPRPD','TSPNDFOOD']].value_counts()
```

Out[298…
```
TSPNDPRPD  TSPNDFOOD
 0.0        0.0         627
           -99.0        251
-88.0      -99.0        249
-99.0       0.0          40
-88.0       0.0          30
dtype: int64
```

In [299…
```
# For the ones where people put 0 for both, going to fill in with 0
df.loc[(df['TSPNDPRPD'] ==0) & (df['TSPNDFOOD'] ==0) &
       (df['PROP_FOODSPEND_HOME'].isna()), 'PROP_FOODSPEND_HOME'] = 0

# If either were -99, put -99 if proportion is still na
df.loc[((df['TSPNDPRPD']==-99) | (df['TSPNDFOOD']==-99)) &
       (df['PROP_FOODSPEND_HOME'].isna()), 'PROP_FOODSPEND_HOME'] = -99

# the rest are -88
df.loc[df['PROP_FOODSPEND_HOME'].isna(), 'PROP_FOODSPEND_HOME'] = -88

# any nulls left?
len(df.loc[df['PROP_FOODSPEND_HOME'].isna(), ['PROP_FOODSPEND_HOME']])
```

Out[299…
```
0
```

In [300…
```
df['PROP_FOODSPEND_HOME'].describe()
```

Out[300…
```
count    77825.000000
mean       -15.689457
std         35.124998
min        -99.000000
25%          0.462366
50%          0.714286
75%          0.877193
max          1.000000
Name: PROP_FOODSPEND_HOME, dtype: float64
```

In [301…
```
# Create column to represent whether household was in a metropolitan statistical
# area or not

df.loc[df['EST_MSA'] > 0, 'IN_METRO_AREA'] = 1
df.loc[df['IN_METRO_AREA'].isna(), 'IN_METRO_AREA'] = 0

df.loc[df['EST_MSA'].isna(), 'EST_MSA'] = 0
```

```
print(df['IN_METRO_AREA'].isna().sum())
df['IN_METRO_AREA'].value_counts()
```

```
0
```

Out[301…
```
0.0    51865
1.0    25960
Name: IN_METRO_AREA, dtype: int64
```

In [302…
```
# convert -99 values to 0 in columns that represent checkboxes that are
# part of a larger multiple choice question. If the checkbox wasn't selected,
# it means No by default.

multichoice_cols = ['SPNDSRC1','SPNDSRC2','SPNDSRC3','SPNDSRC4','SPNDSRC5',
                    'SPNDSRC6','SPNDSRC7','SPNDSRC8','CHNGHOW1','CHNGHOW2',
                    'CHNGHOW3','CHNGHOW4','CHNGHOW5','CHNGHOW6','CHNGHOW7',
                    'CHNGHOW8','CHNGHOW9','CHNGHOW10','CHNGHOW11','CHNGHOW12']

for col in multichoice_cols:
    df.loc[df[col] == -99, col] = 0

df[multichoice_cols].head(5)
```

Out[302…

	SPNDSRC1	SPNDSRC2	SPNDSRC3	SPNDSRC4	SPNDSRC5	SPNDSRC6	SPNDSRC7	SPNDSR
0	-88.0	-88.0	-88.0	-88.0	-88.0	-88.0	-88.0	-8
1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	-88.0	-88.0	-88.0	-88.0	-88.0	-88.0	-88.0	-8
3	0.0	0.0	0.0	1.0	0.0	0.0	1.0	
4	1.0	0.0	0.0	0.0	0.0	0.0	0.0	

In [303…
```
# Convert birth year to age
df['AGE'] = df['TBIRTH_YEAR'].map(lambda x: 2021-x)

df['AGE'].describe()
```

Out[303…
```
count    77825.000000
mean        54.562159
std         15.898557
min         18.000000
25%         42.000000
50%         56.000000
75%         67.000000
max         88.000000
Name: AGE, dtype: float64
```

In [304…
```
df['AGE'].isna().sum()
```

Out[304…
```
0
```

In [305…
```
# Running list of the categorical and numeric column names, both engineered
# and original

cat_cols_orig = ['ANXIOUS','ANYWORK',
                 'CHNGHOW1','CHNGHOW10','CHNGHOW11','CHNGHOW12','CHNGHOW2',
                 'CHNGHOW3','CHNGHOW4','CHNGHOW5','CHNGHOW6','CHNGHOW7','CHNGHOW8',
```

```
            'CHNGHOW9','CURFOODSUF','DELAY','DOWN','EEDUC','EGENDER','EIP',
            'EXPCTLOSS','EXPNS_DIF','FEWRTRANS','FEWRTRIPS','FREEFOOD',
            'HADCOVID','INCOME','INTEREST','LIVQTR','MH_NOTGET','MH_SVCS',
            'MS','NOTGET','PLNDTRIPS','PRESCRIPT','RHISPANIC','RRACE',
            'SNAP_YN','SPNDSRC1','SPNDSRC2','SPNDSRC3','SPNDSRC4','SPNDSRC5',
            'SPNDSRC6','SPNDSRC7','SPNDSRC8','SSA_APPLY','SSA_RECV','TENURE',
            'TW_START','UI_APPLY','UI_RECV','WORRY','WRKLOSS','HLTHINS1',
            'HLTHINS2', 'HLTHINS3', 'HLTHINS4', 'HLTHINS5', 'HLTHINS6',
            'HLTHINS7', 'HLTHINS8', 'ENROLL1', 'ENROLL2', 'ENROLL3', 'EST_MSA']

cat_cols_eng = ['HLTHINS', 'SCHOOL_KIDS', 'IN_METRO_AREA']

cat_cols_orig.sort()

num_cols_orig = ['THHLD_NUMADLT','THHLD_NUMKID', 'TNUM_PS']

num_cols_eng = ['PROP_FOODSPEND_HOME', 'AGE']
```

In [306…

```
# Review cardinality and check for rare labels in categorical columns
for col in cat_cols_orig:
    print(df[col].value_counts(1))
```

```
 1.0     0.351044
 2.0     0.261022
-88.0    0.162082
 4.0     0.129007
 3.0     0.092168
-99.0    0.004677
Name: ANXIOUS, dtype: float64
 1.0     0.568789
 2.0     0.428037
-99.0    0.003174
Name: ANYWORK, dtype: float64
 0.0     0.497783
 1.0     0.448159
-88.0    0.054057
Name: CHNGHOW1, dtype: float64
 0.0     0.884022
 1.0     0.061921
-88.0    0.054057
Name: CHNGHOW10, dtype: float64
 0.0     0.925782
-88.0    0.054057
 1.0     0.020161
Name: CHNGHOW11, dtype: float64
 0.0     0.700405
 1.0     0.245538
-88.0    0.054057
Name: CHNGHOW12, dtype: float64
 0.0     0.725255
 1.0     0.220687
-88.0    0.054057
Name: CHNGHOW2, dtype: float64
 0.0     0.873228
 1.0     0.072714
-88.0    0.054057
Name: CHNGHOW3, dtype: float64
 0.0     0.661022
 1.0     0.284921
-88.0    0.054057
Name: CHNGHOW4, dtype: float64
 0.0     0.913935
-88.0    0.054057
```

```
 1.0     0.032008
Name: CHNGHOW5, dtype: float64
 0.0     0.474513
 1.0     0.471429
-88.0    0.054057
Name: CHNGHOW6, dtype: float64
 0.0     0.867189
 1.0     0.078754
-88.0    0.054057
Name: CHNGHOW7, dtype: float64
 0.0     0.755259
 1.0     0.190684
-88.0    0.054057
Name: CHNGHOW8, dtype: float64
 0.0     0.728519
 1.0     0.217424
-88.0    0.054057
Name: CHNGHOW9, dtype: float64
 1.0     0.653659
 2.0     0.188641
-88.0    0.099852
 3.0     0.041863
 4.0     0.010716
-99.0    0.005268
Name: CURFOODSUF, dtype: float64
 2.0     0.596826
 1.0     0.219004
-88.0    0.178683
-99.0    0.005487
Name: DELAY, dtype: float64
 1.0     0.444896
 2.0     0.244651
-88.0    0.162082
 4.0     0.072689
 3.0     0.069810
-99.0    0.005872
Name: DOWN, dtype: float64
6.0    0.285371
7.0    0.255496
4.0    0.215869
3.0    0.116826
5.0    0.105480
2.0    0.014494
1.0    0.006463
Name: EEDUC, dtype: float64
2.0    0.595259
1.0    0.404741
Name: EGENDER, dtype: float64
 4.0     0.630928
 3.0     0.157032
 1.0     0.097321
 2.0     0.079679
-88.0    0.028423
-99.0    0.006617
Name: EIP, dtype: float64
-88.0    0.738028
 1.0     0.183309
-99.0    0.078664
Name: ENROLL1, dtype: float64
-88.0    0.738028
-99.0    0.239345
 1.0     0.022628
Name: ENROLL2, dtype: float64
-88.0    0.738028
-99.0    0.201015
```

```
 1.0     0.060957
Name: ENROLL3, dtype: float64
0.0        0.666431
47900.0    0.039190
35620.0    0.027125
37980.0    0.026611
14460.0    0.025956
42660.0    0.023746
41860.0    0.023668
31080.0    0.022499
19100.0    0.021587
16980.0    0.021574
26420.0    0.020970
38060.0    0.018362
40140.0    0.015792
19820.0    0.015702
12060.0    0.015676
33100.0    0.015111
Name: EST_MSA, dtype: float64
 2.0     0.850421
 1.0     0.145827
-99.0    0.003752
Name: EXPCTLOSS, dtype: float64
 1.0     0.512894
 2.0     0.209958
 3.0     0.144504
 4.0     0.089444
-88.0    0.037456
-99.0    0.005744
Name: EXPNS_DIF, dtype: float64
 3.0     0.530292
 1.0     0.253736
 2.0     0.114578
-88.0    0.095573
-99.0    0.005821
Name: FEWRTRANS, dtype: float64
 1.0     0.542846
 2.0     0.353344
-88.0    0.095573
-99.0    0.008236
Name: FEWRTRIPS, dtype: float64
 2.0     0.833138
-88.0    0.103206
 1.0     0.059004
-99.0    0.004651
Name: FREEFOOD, dtype: float64
 2.0     0.882236
 1.0     0.109566
 3.0     0.006605
-99.0    0.001593
Name: HADCOVID, dtype: float64
 1.0     0.534623
 2.0     0.262987
-88.0    0.169663
-99.0    0.032727
Name: HLTHINS1, dtype: float64
 2.0     0.537861
 1.0     0.194064
-88.0    0.169663
-99.0    0.098413
Name: HLTHINS2, dtype: float64
 2.0     0.494982
 1.0     0.257861
-88.0    0.169663
-99.0    0.077494
```

```
Name: HLTHINS3, dtype: float64
 2.0     0.626637
-88.0    0.169663
-99.0    0.118484
 1.0     0.085217
Name: HLTHINS4, dtype: float64
 2.0     0.667086
-88.0    0.169663
-99.0    0.124664
 1.0     0.038587
Name: HLTHINS5, dtype: float64
 2.0     0.661934
-88.0    0.169663
-99.0    0.129213
 1.0     0.039190
Name: HLTHINS6, dtype: float64
 2.0     0.685744
-88.0    0.169663
-99.0    0.137347
 1.0     0.007247
Name: HLTHINS7, dtype: float64
 2.0     0.628050
-88.0    0.169663
-99.0    0.163996
 1.0     0.038291
Name: HLTHINS8, dtype: float64
-88.0    0.209586
 6.0     0.140186
 4.0     0.137449
 5.0     0.112278
 3.0     0.083058
 8.0     0.081336
 1.0     0.074655
 7.0     0.069553
 2.0     0.065108
-99.0    0.026791
Name: INCOME, dtype: float64
 1.0     0.423386
 2.0     0.251346
-88.0    0.162082
 3.0     0.081619
 4.0     0.074873
-99.0    0.006695
Name: INTEREST, dtype: float64
 2.0     0.577642
-88.0    0.186341
 3.0     0.063925
 9.0     0.035079
 1.0     0.024684
 6.0     0.022435
 5.0     0.022217
 7.0     0.020071
 8.0     0.019004
 4.0     0.016113
-99.0    0.008969
 10.0    0.003521
Name: LIVQTR, dtype: float64
 2.0     0.730781
-88.0    0.178683
 1.0     0.085551
-99.0    0.004986
Name: MH_NOTGET, dtype: float64
 2.0     0.726656
-88.0    0.178683
 1.0     0.088802
```

```
-99.0    0.005859
Name: MH_SVCS, dtype: float64
 1.0     0.587729
 5.0     0.176833
 3.0     0.152637
 2.0     0.059582
 4.0     0.016948
-99.0    0.006270
Name: MS, dtype: float64
 2.0     0.659582
-88.0    0.178683
 1.0     0.156531
-99.0    0.005204
Name: NOTGET, dtype: float64
 1.0     0.714436
 2.0     0.184851
-88.0    0.095573
-99.0    0.005140
Name: PLNDTRIPS, dtype: float64
 2.0     0.632804
 1.0     0.182885
-88.0    0.178683
-99.0    0.005628
Name: PRESCRIPT, dtype: float64
1.0    0.904953
2.0    0.095047
Name: RHISPANIC, dtype: float64
1.0    0.825609
2.0    0.078073
3.0    0.050434
4.0    0.045885
Name: RRACE, dtype: float64
 2.0     0.823758
-88.0    0.105300
 1.0     0.061972
-99.0    0.008969
Name: SNAP_YN, dtype: float64
 1.0     0.703322
 0.0     0.201105
-88.0    0.095573
Name: SPNDSRC1, dtype: float64
 0.0     0.680308
 1.0     0.224118
-88.0    0.095573
Name: SPNDSRC2, dtype: float64
 0.0     0.699454
 1.0     0.204973
-88.0    0.095573
Name: SPNDSRC3, dtype: float64
 0.0     0.839075
-88.0    0.095573
 1.0     0.065352
Name: SPNDSRC4, dtype: float64
 0.0     0.839640
-88.0    0.095573
 1.0     0.064786
Name: SPNDSRC5, dtype: float64
 0.0     0.734610
 1.0     0.169817
-88.0    0.095573
Name: SPNDSRC6, dtype: float64
 0.0     0.871442
-88.0    0.095573
 1.0     0.032984
Name: SPNDSRC7, dtype: float64
```

```
 0.0     0.863540
-88.0    0.095573
 1.0     0.040887
Name: SPNDSRC8, dtype: float64
 2.0     0.938143
 1.0     0.042955
-88.0    0.010215
-99.0    0.008686
Name: SSA_APPLY, dtype: float64
 2.0     0.656576
 1.0     0.330960
-88.0    0.006669
-99.0    0.005795
Name: SSA_RECV, dtype: float64
 2.0     0.395708
 1.0     0.229978
-88.0    0.186341
 3.0     0.171770
 4.0     0.010434
-99.0    0.005769
Name: TENURE, dtype: float64
 1.0     0.412425
 2.0     0.374892
 3.0     0.164896
-99.0    0.047787
Name: TW_START, dtype: float64
 2.0     0.842094
 1.0     0.152586
-99.0    0.002788
-88.0    0.002531
Name: UI_APPLY, dtype: float64
-88.0    0.847941
 1.0     0.121825
 2.0     0.027857
-99.0    0.002377
Name: UI_RECV, dtype: float64
 1.0     0.428898
 2.0     0.238818
-88.0    0.162082
 4.0     0.088930
 3.0     0.075130
-99.0    0.006142
Name: WORRY, dtype: float64
 2.0     0.626611
 1.0     0.371012
-99.0    0.002377
Name: WRKLOSS, dtype: float64
```

In [307…

```
# review descriptive statistics for numeric columns
for col in num_cols_orig:
    print(df[col].isna().sum())
    print(df[col].describe())
```

```
0
count    77825.000000
mean         2.122981
std          0.966851
min          1.000000
25%          2.000000
50%          2.000000
75%          2.000000
max         10.000000
Name: THHLD_NUMADLT, dtype: float64
0
count    77825.000000
```

```
mean         0.588024
std          1.010032
min          0.000000
25%          0.000000
50%          0.000000
75%          1.000000
max          5.000000
Name: THHLD_NUMKID, dtype: float64
0
count    77825.000000
mean       -19.387189
std         37.068353
min        -99.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          3.000000
Name: TNUM_PS, dtype: float64
```

Of the columns I initially had listed as categorical, I used the value counts above, combined with reviewing the meanings of each label in the data dictionary, to make further decisions on each column.

- Many of these columns are almost already OHE. There are already only 2 choices, mostly Yes/No answers, but a few others such as gender Male/Female.
  - Many of the Yes/No questions actually have Yes encoded as 1 and No encoded as 2, so I'm going to mass replace those with 0 instead of 2 to be more consistent with how OHE typically works.
  - A few can be turned into OHE columns if I combine rare labels.

- A few categorical columns are suited to be left as ordinals and treated like numbers. These columns are:
  - `ANXIOUS`, `DOWN`, `INTEREST` (lack of), and `WORRY`: as value increases, frequency of experiencing these feelings increases.
  - `CURFOODSUF`: as value increases, food insecurity increases / food suffiency decreases.
  - `EEDUC`: as value increases, education level represented by years of schooling required to obtain that level increases.
  - `EXPNS_DIF`: as value increases, difficulty with expenses increases
  - `INCOME`: as value increases, pre-tax income range increases

- Some categorical columns do need to be OHE because they have more than two choices that can't be combined without losing information.

- I will drop `UI_RECV` because the vast majority of its values are -88 missing, so it has little to contribute in terms of actual answers the respondents provided.

Almost all of the columns have -99 values that represent the respondent skipped the question, and -88 values that are simply missing with no explanation. Before accepting that I need to make 2 separate columns for every one of these (expanding my feature space by a lot) I'm going to map them using missingno and see if I can identify patterns. Maybe values that are -88 across multiple columns can be bundled together in fewer missing indicator columns.

Of the numerical columns, I will turn `TNUM_PS` into a Yes/No categorical column, since it appears most households have no one who was planning on post-secondary education in fall 2021. I don't think the number of people will make a big difference.

In [308…

```
# for columns that really only have 2 answers, most are 1 and 2. Leave 1 as Yes
# and convert the 2s to 0s for Nos or second value.

conv_2_to_0 = ['ANYWORK', 'DELAY', 'EGENDER', 'EXPCTLOSS', 'FEWRTRIPS',
               'FREEFOOD', 'HADCOVID', 'MH_NOTGET', 'MH_SVCS', 'NOTGET',
               'PLNDTRIPS','PRESCRIPT', 'SNAP_YN', 'SSA_APPLY', 'SSA_RECV',
               'UI_APPLY', 'WRKLOSS', 'HLTHINS1', 'HLTHINS2', 'HLTHINS3',
               'HLTHINS4', 'HLTHINS5', 'HLTHINS6', 'HLTHINS7', 'HLTHINS8']

for col in conv_2_to_0:
    df.loc[df[col] == 2, col] = 0
    print(df[col].value_counts())
```

```
 1.0     44266
 0.0     33312
-99.0      247
Name: ANYWORK, dtype: int64
 0.0     46448
 1.0     17044
-88.0    13906
-99.0      427
Name: DELAY, dtype: int64
0.0    46326
1.0    31499
Name: EGENDER, dtype: int64
 0.0     66184
 1.0     11349
-99.0      292
Name: EXPCTLOSS, dtype: int64
 1.0     42247
 0.0     27499
-88.0     7438
-99.0      641
Name: FEWRTRIPS, dtype: int64
 0.0     64839
-88.0     8032
 1.0      4592
-99.0      362
Name: FREEFOOD, dtype: int64
 0.0     68660
 1.0      8527
 3.0       514
-99.0      124
Name: HADCOVID, dtype: int64
 0.0     56873
-88.0    13906
 1.0      6658
-99.0      388
Name: MH_NOTGET, dtype: int64
 0.0     56552
-88.0    13906
 1.0      6911
-99.0      456
Name: MH_SVCS, dtype: int64
 0.0     51332
-88.0    13906
 1.0     12182
-99.0      405
Name: NOTGET, dtype: int64
```

```
 1.0     55601
 0.0     14386
-88.0     7438
-99.0      400
Name: PLNDTRIPS, dtype: int64
 0.0     49248
 1.0     14233
-88.0    13906
-99.0      438
Name: PRESCRIPT, dtype: int64
 0.0     64109
-88.0     8195
 1.0      4823
-99.0      698
Name: SNAP_YN, dtype: int64
 0.0     73011
 1.0      3343
-88.0      795
-99.0      676
Name: SSA_APPLY, dtype: int64
 0.0     51098
 1.0     25757
-88.0      519
-99.0      451
Name: SSA_RECV, dtype: int64
 0.0     65536
 1.0     11875
-99.0      217
-88.0      197
Name: UI_APPLY, dtype: int64
 0.0     48766
 1.0     28874
-99.0      185
Name: WRKLOSS, dtype: int64
 1.0     41607
 0.0     20467
-88.0    13204
-99.0     2547
Name: HLTHINS1, dtype: int64
 0.0     41859
 1.0     15103
-88.0    13204
-99.0     7659
Name: HLTHINS2, dtype: int64
 0.0     38522
 1.0     20068
-88.0    13204
-99.0     6031
Name: HLTHINS3, dtype: int64
 0.0     48768
-88.0    13204
-99.0     9221
 1.0      6632
Name: HLTHINS4, dtype: int64
 0.0     51916
-88.0    13204
-99.0     9702
 1.0      3003
Name: HLTHINS5, dtype: int64
 0.0     51515
-88.0    13204
-99.0    10056
 1.0      3050
Name: HLTHINS6, dtype: int64
 0.0     53368
```

```
-88.0    13204
-99.0    10689
 1.0       564
Name: HLTHINS7, dtype: int64
 0.0     48878
-88.0    13204
-99.0    12763
 1.0      2980
Name: HLTHINS8, dtype: int64
```

In [309…
```
# OHE HADCOVID by combining rare label 3 - Not Sure with -99 No answer
df.loc[df['HADCOVID']==3, 'HADCOVID'] = -99
df['HADCOVID'].value_counts()
```

Out[309…
```
 0.0     68660
 1.0      8527
-99.0      638
Name: HADCOVID, dtype: int64
```

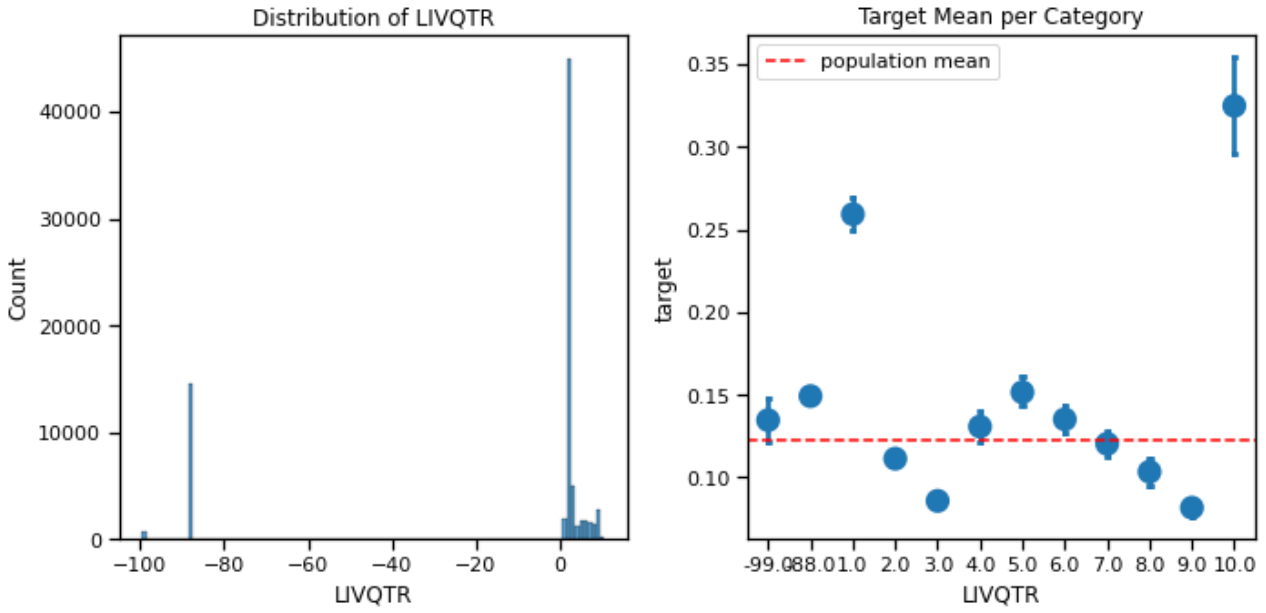
In [310…
```
# Make RHISPANIC consistent with 0=No, 1=Yes
df.loc[df['RHISPANIC']==1, 'RHISPANIC'] = 0
df.loc[df['RHISPANIC']==2, 'RHISPANIC'] = 1
df['RHISPANIC'].value_counts()
```

Out[310…
```
0.0    70428
1.0     7397
Name: RHISPANIC, dtype: int64
```

In [311…
```
# TW_Start labels 2 and 3 are both no telework, so converting to 0
df.loc[(df['TW_START']==2)|(df['TW_START']==3), 'TW_START'] = 0
df['TW_START'].value_counts()
```

Out[311…
```
 0.0     42009
 1.0     32097
-99.0     3719
Name: TW_START, dtype: int64
```

In [312…
```
dstools.explore_data_catbin(['LIVQTR'], df, 'target')
```



In [313…
```
# LIVQTR will need to be OHE, but I can combine some labels and not lose
# much information
```

```
df['LIVQTR'].value_counts()
```

Out[313…
```
 2.0     44955
-88.0    14502
 3.0      4975
 9.0      2730
 1.0      1921
 6.0      1746
 5.0      1729
 7.0      1562
 8.0      1479
 4.0      1254
-99.0      698
 10.0      274
Name: LIVQTR, dtype: int64
```

Category labels from the data dictionary:

1. A mobile home
2. A one-family house detached from any other house
3. A one-family house attached to one or more houses
4. A building with 2 apartments
5. A building with 3 or 4 apartment
6. A building with 5 to 9 apartments
7. A building with 10 to 19 apartments
8. A building with 20 to 49 apartments
9. A building with 50 or more apartments
10. Boat, RV, van, etc.

I will combine these 10 categories into 4:

- 1) Mobile home
- 2 and 3 -> 2) House
- 4 through 9 -> 3) Apartment
- 10 -> 4) Boat, RV, van

In [314…
```
# Combine some of the categories in my engineered feature LIVQTR_2
df.loc[df['LIVQTR']==1, 'LIVQTR_2'] = 1
df.loc[df['LIVQTR'].isin([2, 3]), 'LIVQTR_2'] = 2
df.loc[df['LIVQTR'].isin([4,5,6,7,8,9]), 'LIVQTR_2'] = 3
df.loc[df['LIVQTR']==10, 'LIVQTR_2'] = 4
```

In [315…
```
df['LIVQTR_2'].value_counts()
```

Out[315…
```
2.0    49930
3.0    10500
1.0     1921
4.0      274
Name: LIVQTR_2, dtype: int64
```

In [316…
```
df['LIVQTR_2'].isna().sum()
```

Out[316…
```
15200
```

In [317…
```
# Fill in -88 and -99 values from original feature
```

```
df.loc[df['LIVQTR']==-88, 'LIVQTR_2'] = -88
df.loc[df['LIVQTR']==-99, 'LIVQTR_2'] = -99
print(df['LIVQTR_2'].value_counts())
df['LIVQTR_2'].isna().sum()
```

```
 2.0     49930
-88.0    14502
 3.0     10500
 1.0      1921
-99.0      698
 4.0       274
Name: LIVQTR_2, dtype: int64
```

Out[317… 0

In [318…
```
# Combine a few rare Married Status labels
# Adding 3 - divorced and 4 - separated into 2 - Widowed (all sound single now
# but used to be married)
# 1 is married, and 5 is never married

df.loc[df['MS']==1, 'MS_2'] = 1
df.loc[df['MS'].isin([2, 3, 4]), 'MS_2'] = 2
df.loc[df['MS']==5, 'MS_2'] = 3
df.loc[df['MS']==-88, 'MS_2'] = -88
df.loc[df['MS']==-99, 'MS_2'] = -99
print(df['MS_2'].value_counts())
df['MS_2'].isna().sum()
```

```
 1.0     45740
 2.0     17835
 3.0     13762
-99.0      488
Name: MS_2, dtype: int64
```

Out[318… 0

In [319…
```
# Review TNUM_PS
df['TNUM_PS'].value_counts()
```

Out[319…
```
 0.0     45969
-88.0    15652
 1.0     10678
 2.0      3138
-99.0     1525
 3.0       863
Name: TNUM_PS, dtype: int64
```

In [320…
```
# These represent the number of people in the household who plan to go to
# post-secondary classes in 2021. Since there aren't a lot of 2 and 3 values,
# will combine them into 1 so it's just a Yes have people who plan to take
# PS classes, or No have none in the household.

df['TNUM_PS_2'] = df['TNUM_PS']
df.loc[df['TNUM_PS_2'].isin([2, 3]), 'TNUM_PS_2'] = 1

df['TNUM_PS_2'].value_counts()
```

Out[320…
```
 0.0     45969
-88.0    15652
 1.0     14679
-99.0     1525
Name: TNUM_PS_2, dtype: int64
```

In [321…
```
# drop UI_RECV from cat_cols, as majority of values are -88 missing
df['UI_RECV'].value_counts(normalize=True)
```

Out[321…
```
-88.0    0.847941
 1.0     0.121825
 2.0     0.027857
-99.0    0.002377
Name: UI_RECV, dtype: float64
```

In [322…
```
# Update running lists of categorical and numeric columns
cat_cols_orig.remove('UI_RECV')

cat_cols_eng.append('MS_2')
cat_cols_eng.append('LIVQTR_2')
cat_cols_eng.append('TNUM_PS_2')
```

Review Locations of -88 Missing

There are some sets of predictor columns that seem to have the same number of -88 values, from looking at the value counts. But the data dictionary doesn't provide much info on what -88 represents.

I will try to visualize these to see if there are patterns. Perhaps I can use combine missing indicator columns.

In [323…
```
# original predictor columns, ordered by when questions were presented in
# the questionnaire

q_order = ['TBIRTH_YEAR','EGENDER','RHISPANIC','RRACE','EEDUC','MS',
           'THHLD_NUMKID','THHLD_NUMADLT','RECVDVACC','HADCOVID',
           'WRKLOSS','EXPCTLOSS','ANYWORK','TW_START','UI_APPLY','UI_RECV',
           'SSA_RECV','SSA_APPLY','EIP','EXPNS_DIF','SPNDSRC1',
           'SPNDSRC2','SPNDSRC3','SPNDSRC4','SPNDSRC5','SPNDSRC6','SPNDSRC7',
           'SPNDSRC8','FEWRTRIPS','FEWRTRANS','PLNDTRIPS','CURFOODSUF',
           'FREEFOOD','SNAP_YN','TSPNDFOOD','TSPNDPRPD','ANXIOUS','WORRY',
           'INTEREST','DOWN','HLTHINS1', 'HLTHINS2', 'HLTHINS3', 'HLTHINS4',
           'HLTHINS5', 'HLTHINS6', 'HLTHINS7', 'HLTHINS8','DELAY','NOTGET',
           'PRESCRIPT','MH_SVCS','MH_NOTGET','TENURE','LIVQTR','ENROLL1',
           'ENROLL2','ENROLL3','TNUM_PS','INCOME']
```

In [324…
```
# create a copy of the columns so I can replace -88 with nulls and visualize
nan_viz = df[q_order].copy()
print(list(nan_viz.isna().sum()))
```

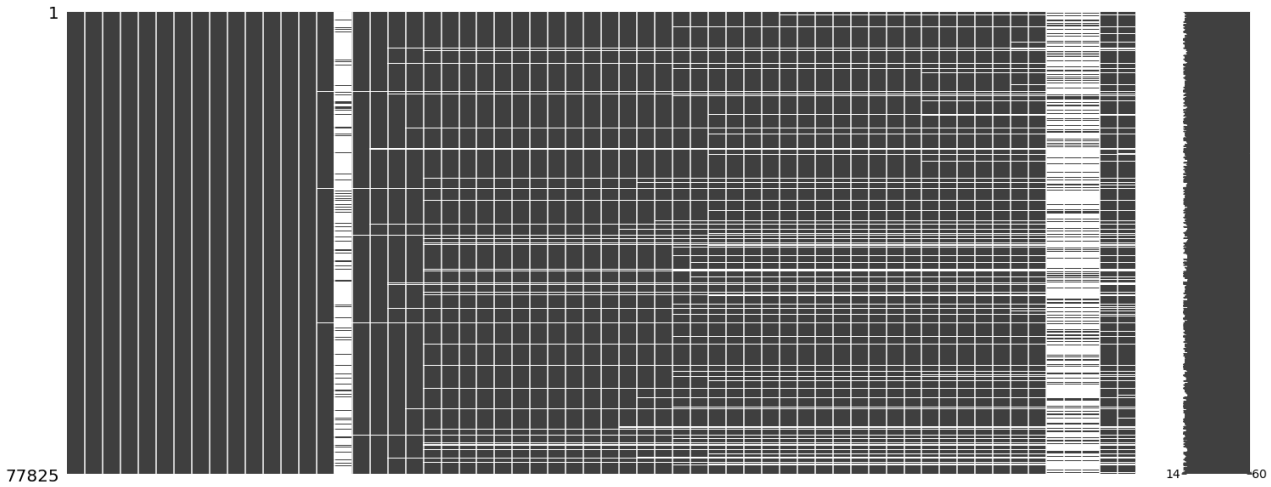
```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0]
```

In [325…
```
# replace -88s with nans
nan_viz.replace(-88, np.nan, inplace=True)
print(list(nan_viz.isna().sum()))
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 197, 65991, 519, 795, 2212, 2915, 743
8, 7438, 7438, 7438, 7438, 7438, 7438, 7438, 7438, 7438, 7438, 7771, 8032, 8195,
10775, 11054, 12614, 12614, 12614, 12614, 13204, 13204, 13204, 13204, 13204, 132
04, 13204, 13204, 13906, 13906, 13906, 13906, 13906, 14502, 14502, 57437, 57437,
57437, 15652, 16311]
```

In [326…
```
# visualize where the -88 values are
```

```
missingno.matrix(nan_viz);
```



Hmm, when I look at them in the order the questions were presented, I see that with the exception of a few columns that have a lot of nans, once missing answers start showing up, the rest of that record has missing answers.

I suspect that the -88 Missing placeholder indicates that the respondent exited the questionnaire prematurely, and Census kept the incomplete answers in the sample.

After re-reviewing the technical documentation, I found this paragraph buried towards the end:

> Responses are made up of complete interviews and sufficient partial interviews. A sufficient partial interview is an incomplete interview in which the household or person answered enough of the questionnaire to be considered a complete interview. Some remaining questions may have been edited or imputed to fill in missing values. Insufficient partial interviews are considered to be nonrespondents.

But there are a few columns that stand out as having a lot of -88 values and aren't following this pattern. I would like to look further into those before deciding how to proceed.

In [327…
```
# look at sum of missing values for each question, in order they were asked
# Which columns don't follow the steady increase?
missing_sum = nan_viz.isna().sum(axis=0)
missing_sum.loc[q_order]
```

Out[327…
```
TBIRTH_YEAR          0
EGENDER              0
RHISPANIC            0
RRACE                0
EEDUC                0
MS                   0
THHLD_NUMKID         0
THHLD_NUMADLT        0
RECVDVACC            0
HADCOVID             0
WRKLOSS              0
EXPCTLOSS            0
ANYWORK              0
TW_START             0
UI_APPLY           197
```

```
UI_RECV          65991
SSA_RECV           519
SSA_APPLY          795
EIP               2212
EXPNS_DIF         2915
SPNDSRC1          7438
SPNDSRC2          7438
SPNDSRC3          7438
SPNDSRC4          7438
SPNDSRC5          7438
SPNDSRC6          7438
SPNDSRC7          7438
SPNDSRC8          7438
FEWRTRIPS         7438
FEWRTRANS         7438
PLNDTRIPS         7438
CURFOODSUF        7771
FREEFOOD          8032
SNAP_YN           8195
TSPNDFOOD        10775
TSPNDPRPD        11054
ANXIOUS          12614
WORRY            12614
INTEREST         12614
DOWN             12614
HLTHINS1         13204
HLTHINS2         13204
HLTHINS3         13204
HLTHINS4         13204
HLTHINS5         13204
HLTHINS6         13204
HLTHINS7         13204
HLTHINS8         13204
DELAY            13906
NOTGET           13906
PRESCRIPT        13906
MH_SVCS          13906
MH_NOTGET        13906
TENURE           14502
LIVQTR           14502
ENROLL1          57437
ENROLL2          57437
ENROLL3          57437
TNUM_PS          15652
INCOME           16311
dtype: int64
```

It looks like the four columns that don't fit the pattern are `UI_RECV`, `ENROLL1`, `ENROLL2`, and `ENROLL3`.

I already confirmed above that the school enrollment questions which had a bunch of -88 values were mostly people who reported 0 kids in the household, so I don't need to confirm that again.

I had also already removed `UI_RECV` from my list of columns to explore further because if the high number of -88 values, but I'll look at it again here to make sure what I see fits my hypothesis about the meaning of the -88 values.

In [328…

```
# UI_APPLY might be related to UI_RECV. Let's look at both and see.
df['UI_APPLY'].value_counts()
```

Out[328…

```
0.0     65536
1.0     11875
```

```
-99.0      217
-88.0      197
Name: UI_APPLY, dtype: int64
```

In [329…
```
df['UI_RECV'].value_counts()
```

Out[329…
```
-88.0    65991
 1.0      9481
 2.0      2168
-99.0      185
Name: UI_RECV, dtype: int64
```

In [330…
```
# What is the sum of people who answered No (0) or didn't answer 'UI_APPLY'?
print(65536 + 217 + 197)
```

```
65950
```

That's pretty darn close. I'm thinking `UI_RECV` was only presented to people who said they actually applied for unemployment benefits, so most people weren't shown it.

Strategy for Incomplete Questionnaires

I found the section of the technical documentation that says there are incomplete questionnaires, and the patterns I can see in the responses matches that. Now I need to decide whether to simply drop these incomplete answers and use only the complete ones, or account for them in some way.

To decide, I'll take a look at the class distribution for incomplete versus complete questionnaires and see if there appears to be a difference.

In [331…
```
# drop the columns I know weren't shown to everyone
q_order.remove('ENROLL1')
q_order.remove('ENROLL2')
q_order.remove('ENROLL3')
q_order.remove('UI_RECV')
```

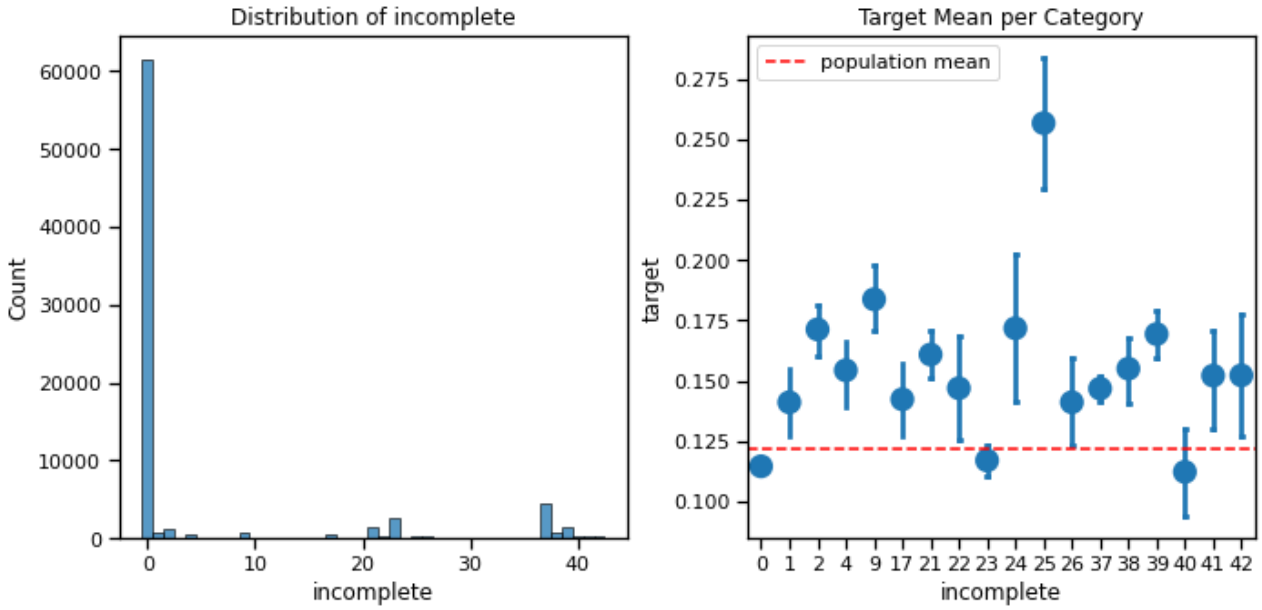
In [332…
```
# populate incomplete column with sum of -88 values in top-level q columns
def sum_neg88(row):
    missing = row.isin([-88]).sum(axis=0)
    return missing

df['incomplete'] = df[q_order].apply(sum_neg88, axis=1)

df['incomplete']
```

Out[332…
```
0        37
1         0
2        39
3         0
4         0
         ..
77820     0
77821     0
77822     0
77823     0
77824     0
Name: incomplete, Length: 77825, dtype: int64
```

In [333…
```
# Add 'incomplete' to the list of engineered cat_cols
```

```
cat_cols_eng.append('incomplete')
```

In [334…
```
# plot number of incomplete questionnaires against target
dstools.explore_data_catbin(['incomplete'], df, 'target')
```



Interesting, so there is some variability based on number of questions.

I will keep the `incomplete` column that has the count of complete questions for each respondent (i.e. how many questions were left when they exited) but also create a binary column to indicate whether they finished the whole thing or not.

To make it easy to get rid of the -88 values I will set them equal to the 'first', or lowest value in the column, and when I use sklearn's OneHotEncoder, I will drop the first column.

I will convert -88 to -2 so it will be 'first' and will convert -99, which represents someone viewing but skipping a particular question, as -1.

In [335…
```
# Create inc_binary column
df.loc[df['incomplete'] == 0, 'inc_binary'] = 0
df.loc[df['incomplete'] > 0, 'inc_binary'] = 1

df['inc_binary'].value_counts()
```

Out[335…
```
0.0    61514
1.0    16311
Name: inc_binary, dtype: int64
```

In [336…
```
df['inc_binary'].isna().sum()
```

Out[336… 0

I'll do the same thing for -99 values. I'll create a numeric column to indicate the number of skipped questions based on -99, and will also OHE the -99 category for each categorical column so I can try modeling both ways and see which does better.

In [337…
```
# populate skipped column with sum of -88 values in top-level q columns
def sum_neg99(row):
```

```
    missing = row.isin([-99]).sum(axis=0)
    return missing

df['skipped'] = df[q_order].apply(sum_neg99, axis=1)

df['skipped'].value_counts()
```

Out[337…

```
0     54560
1      8389
7      4782
2      2647
6      2188
3      1344
5      1261
4       920
8       758
9       345
10      160
11       70
12       48
13       34
14       32
23       27
21       24
29       23
18       20
17       18
15       17
16       16
19       16
20       15
25       15
22       15
26       12
27        9
30        8
24        7
28        7
39        6
34        6
32        6
31        4
33        3
35        3
36        3
38        3
37        2
40        2
Name: skipped, dtype: int64
```

In [338…

```
# add new columns to lists
num_cols_eng.append('incomplete')
num_cols_eng.append('skipped')
cat_cols_eng.append('inc_binary')
```

In [339…

```
# replace -88 with -2 and -99 with -1 for all cat cols
# I will still need to deal with some missing values in num cols

for col in cat_cols_orig + cat_cols_eng:
    df.loc[df[col] == -88, col] = -2
    df.loc[df[col] == -99, col] = -1
```

```
# check for remaining -88 and -99 values
print(list(df[cat_cols_orig + cat_cols_eng].isin([-88, -99]).sum(axis=0)))
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

In [340…
```
# take a look at which num cols have -88 or -99 values
df[num_cols_eng + num_cols_orig].isin([-88, -99]).sum(axis=0)
```

Out[340…
```
PROP_FOODSPEND_HOME    14015
AGE                        0
incomplete                 0
skipped                    0
THHLD_NUMADLT              0
THHLD_NUMKID               0
TNUM_PS                17177
dtype: int64
```

I will need to remember to do something else with these numerical values when I prepare the data for modeling, as they will need to be replaced with something. But for now, so I can chart the columns against the target, I will see if it's safe to assign -2 and -1 to my placeholders. I want to set them on a scale that will work well with the rest of the values when I chart them on a continuous x axis, but also be easy to single out for replacement with something else later.

In [341…
```
# are there any values less than 0 other than -88 and -99?
df.loc[df['PROP_FOODSPEND_HOME'] < 0, ['PROP_FOODSPEND_HOME']].value_counts()
```

Out[341…
```
PROP_FOODSPEND_HOME
-88.0                  10805
-99.0                   3210
dtype: int64
```

In [342…
```
# are there any values less than 0 other than -88 and -99?
df.loc[df['TNUM_PS'] < 0, ['TNUM_PS']].value_counts()
```

Out[342…
```
TNUM_PS
-88.0      15652
-99.0       1525
dtype: int64
```

In [343…
```
# Both columns look fine to replace with -2 and -1
for col in ['TNUM_PS', 'PROP_FOODSPEND_HOME']:
    df.loc[df[col] == -88, col] = -2
    df.loc[df[col] == -99, col] = -1
```

EXPLORE

Now I have my predictors fairly cleaned up: some rare labels combined, a better understanding of the -99 (now -1) and -88 (now -2) missing indicators, and a few features engineered.

I will visualize each variable's distribution and relationship to the target to see if any need further engineering.

In [127…
```
rerun_viz = False
```

In [119…
```
# reset my lists of categorical and numeric columns after engineering
```

```
# I'll have a new list for potentially ordinal columns,
# because they will be treated differently in preprocessing

# These are the columns that only have one version
cat_cols = ['ANYWORK','CHNGHOW1','CHNGHOW10','CHNGHOW11','CHNGHOW12',
            'CHNGHOW2','CHNGHOW3','CHNGHOW4','CHNGHOW5','CHNGHOW6',
            'CHNGHOW7','CHNGHOW8','CHNGHOW9','DELAY','EGENDER','EIP',
            'EXPCTLOSS','FEWRTRANS','FEWRTRIPS','FREEFOOD','HADCOVID',
            'MH_NOTGET', 'MH_SVCS','NOTGET','PLNDTRIPS','PRESCRIPT','RHISPANIC',
            'RRACE','SNAP_YN','SPNDSRC1','SPNDSRC2','SPNDSRC3',
            'SPNDSRC4','SPNDSRC5','SPNDSRC6','SPNDSRC7','SPNDSRC8',
            'SSA_APPLY','SSA_RECV','TENURE','TW_START','UI_APPLY','WRKLOSS']

# I engineered versions of these columns, but these are the originals
# They have more rare labels, and provide more detail than the engineered
# versions
cat_cols_v1 = ['HLTHINS1', 'HLTHINS2', 'HLTHINS3', 'HLTHINS4', 'HLTHINS5',
               'HLTHINS6', 'HLTHINS7', 'HLTHINS8', 'ENROLL1', 'ENROLL2',
               'ENROLL3', 'EST_MSA', 'LIVQTR', 'MS']

# These are the engineered versions of the columns that I can try if the
# original versions don't perform well
cat_cols_v2 = ['HLTHINS','inc_binary','IN_METRO_AREA', 'SCHOOL_KIDS',
               'LIVQTR_2', 'MS_2', 'TNUM_PS_2']

# These are the columns that are categorical, but I think can be treated as
# numbers
ord_cols = ['ANXIOUS','CURFOODSUF','DOWN','EEDUC','EXPNS_DIF','INCOME',
            'INTEREST','WORRY']

# Numeric columns
# incomplete is listed here because I'm going to include it either way
num_cols = ['THHLD_NUMADLT','THHLD_NUMKID','PROP_FOODSPEND_HOME','AGE']

# I'll replace incomplete with inc_binary, which is a categorical column
num_cols_v1 = ['incomplete', 'TNUM_PS']

# I can try out this column instead of OHE -99 as separate columns
num_cols_v2 = ['skipped']
```

In [497…

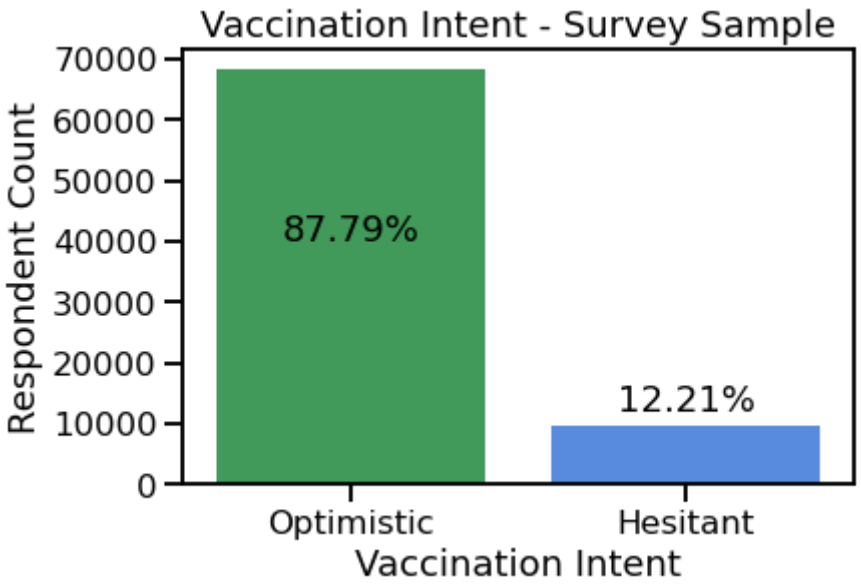
```
# Map v1 columns to v2 columns so I know which versions are connected
v1_v2_map = [{'v1': ['HLTHINS1', 'HLTHINS2', 'HLTHINS3', 'HLTHINS4',
                     'HLTHINS5', 'HLTHINS6', 'HLTHINS7', 'HLTHINS8'],
              'v2': ['HLTHINS'], 'priority': 'v1'},
             {'v1': ['ENROLL1', 'ENROLL2', 'ENROLL3'], 'v2': ['SCHOOL_KIDS'],
              'priority': 'v2'},
             {'v1': ['EST_MSA'], 'v2': ['IN_METRO_AREA'], 'priority': 'v1'},
             {'v1': ['LIVQTR'], 'v2': ['LIVQTR_2'], 'priority': 'v1'},
             {'v1': ['MS'], 'v2': ['MS_2'], 'priority': 'v1'},
             {'v1': ['TNUM_PS'], 'v2': ['TNUM_PS_2'], 'priority': 'v1'},
             {'v1': ['incomplete'], 'v2': ['inc_binary'], 'priority': 'v1'}

]
```

Target

In [616…

```
# Visualize Target Distribution
# some code adapted from:
# https://github.com/mwaskom/seaborn/issues/1027
```

```
with sns.plotting_context(context='talk'):

    colors = sns.color_palette(['#34a853', '#4285f4'])

    fig, ax = plt.subplots()

    sns.barplot(x='target', y='target', data=df,
                estimator=lambda x: len(x), ci=68, ax=ax, palette=colors)

    ax.set_title('Vaccination Intent - Survey Sample')
    ax.set_ylabel('Respondent Count')
    ax.set_xlabel('Vaccination Intent')

    ax.text(x=0, y=40000, s=f"{opt_per}%", ha='center')
    ax.text(x=1, y=12000, s=f"{pes_per}%", ha='center')
    ax.set_xticklabels(['Optimistic', 'Hesitant']);
```



In [344…
```
opt_per = np.round(df['target'].value_counts(1).loc[0.0] * 100, 2)
pes_per = np.round(df['target'].value_counts(1).loc[1.0] * 100, 2)
print(opt_per, pes_per)
```

```
87.79 12.21
```

In [347…
```
# Check whether numeric columns should be visulized separately or with cat_cols
# depending on how many unique values they have
for col in num_cols + num_cols_v1 + num_cols_v2:
    print(f"Column {col} has {len(df[col].value_counts())} values.")
```

```
Column THHLD_NUMADLT has 10 values.
Column THHLD_NUMKID has 6 values.
Column PROP_FOODSPEND_HOME has 2097 values.
Column AGE has 71 values.
Column incomplete has 18 values.
Column skipped has 41 values.
```

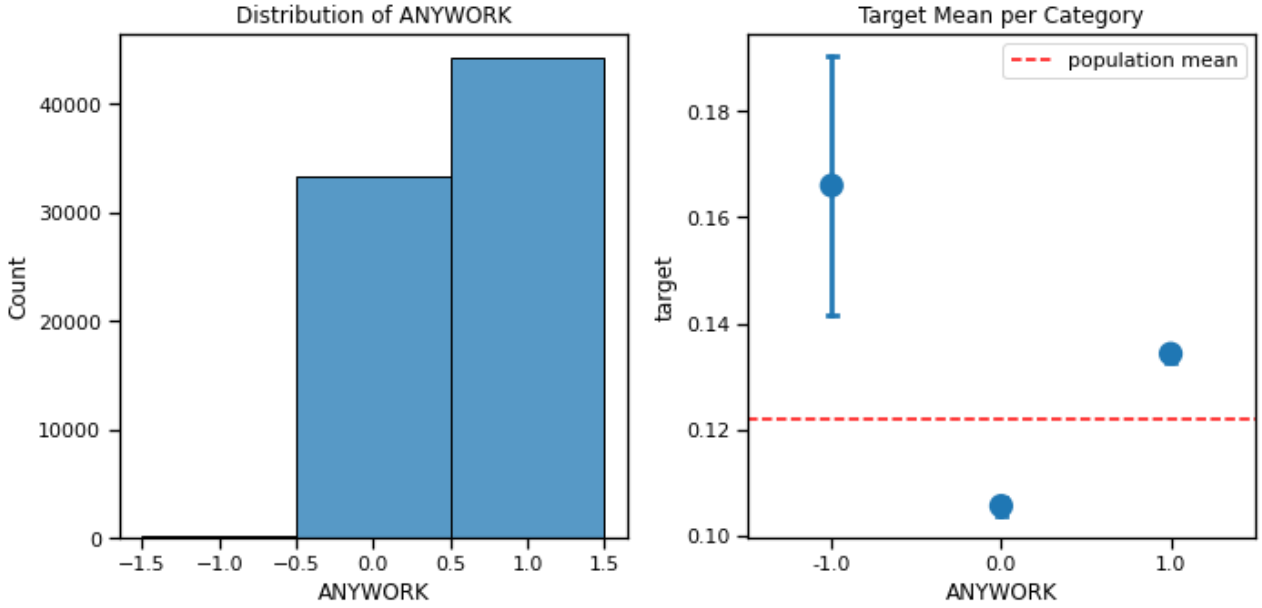
I really only have 4 numeric values in terms of how I want to visualize them: `PROP_FOODSPEND_HOME`, `AGE`, `incomplete`, and `skipped`. The other 3 are numbers but are discrete and have few values.

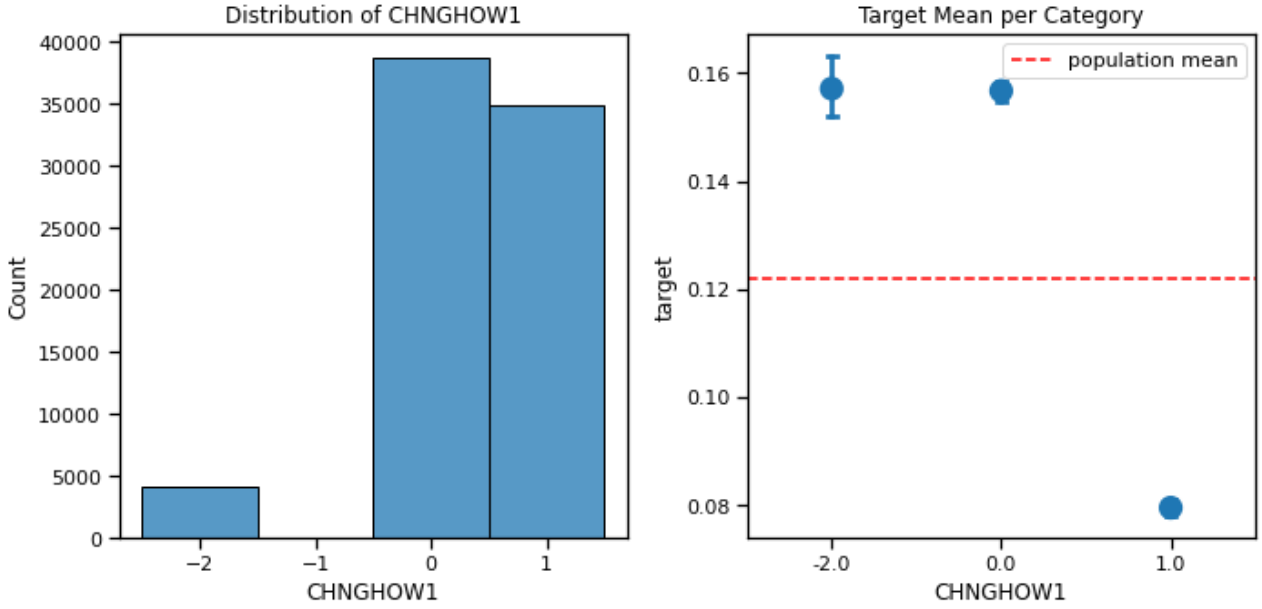
Categorical Features

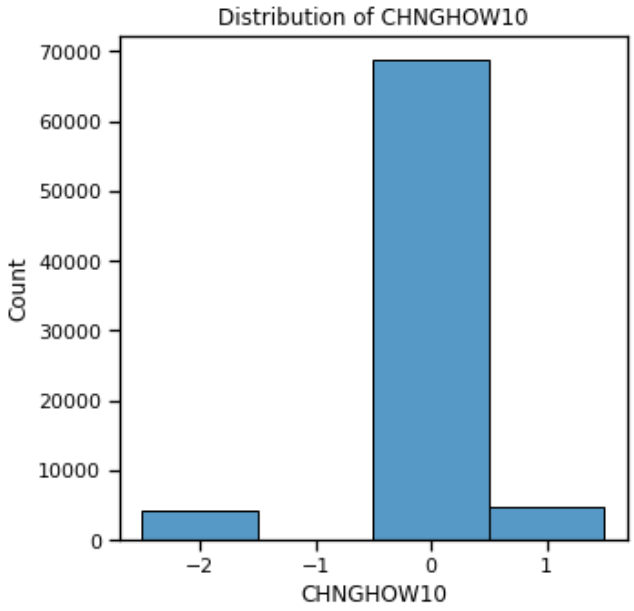
In [348…

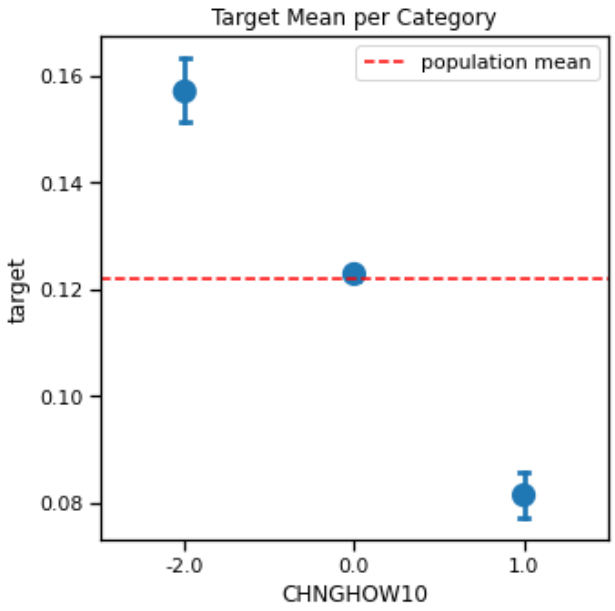
```
cats_viz = cat_cols + cat_cols_v1 + cat_cols_v2

if rerun_viz:
    dstools.explore_data_catbin(cats_viz, df, 'target')
```

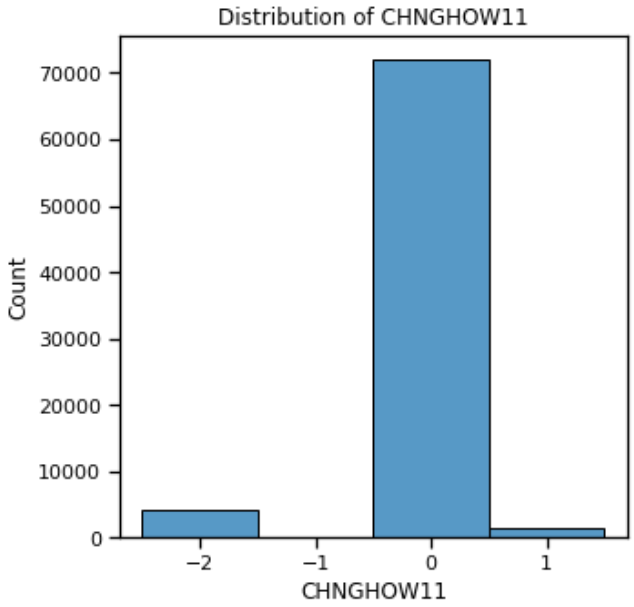
```
/Users/jessicamiles/Documents/Flatiron_Data_Science/my_projects/git_repos/ushps-covidvax/dstools/__init__.py:343: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max_open_warning`).
  fig, [ax1, ax2] = plt.subplots(figsize=(10, 5), nrows=1, ncols=2)
```

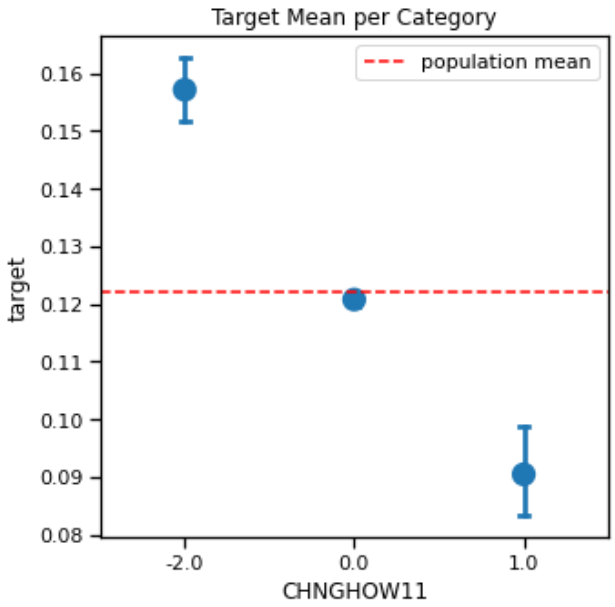


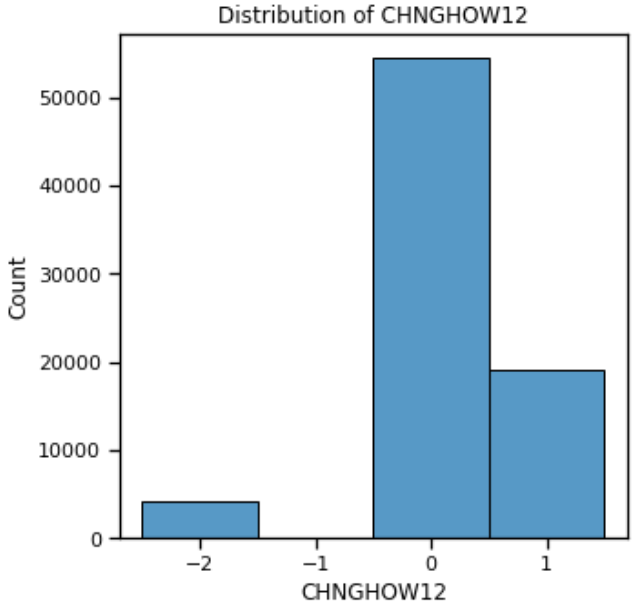


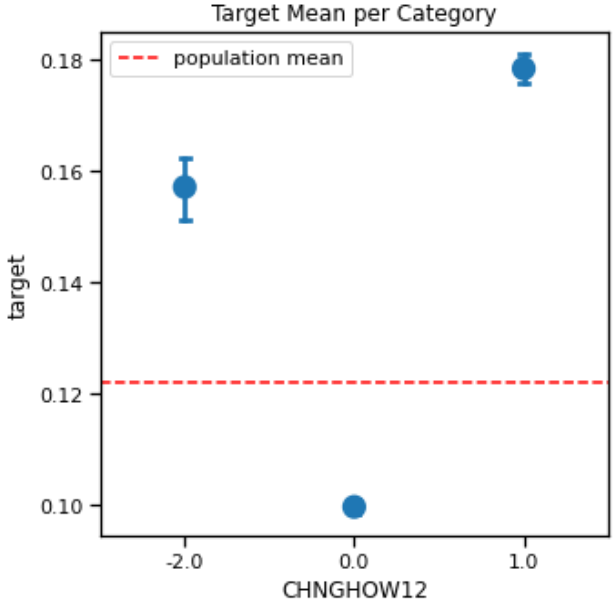

Distribution of CHNGHOW10
Count
CHNGHOW10
70000
60000
50000
40000
30000
20000
10000
0
−2
−1
0
1

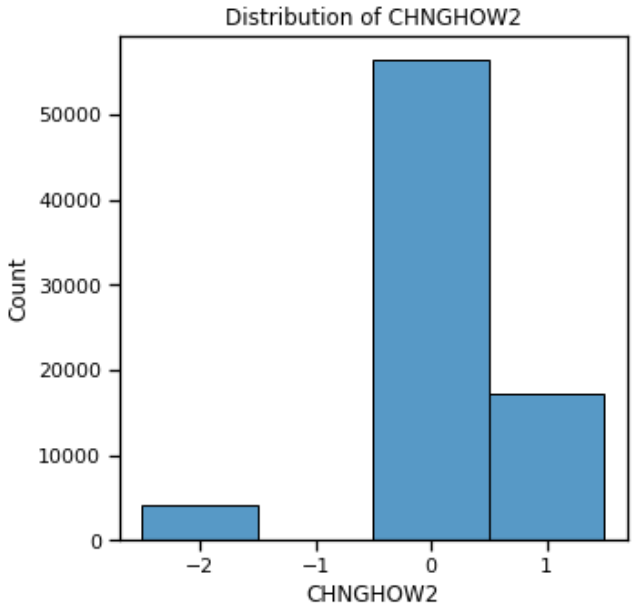

Target Mean per Category
population mean
target
CHNGHOW10
0.16
0.14
0.12
0.10
0.08
-2.0
0.0
1.0

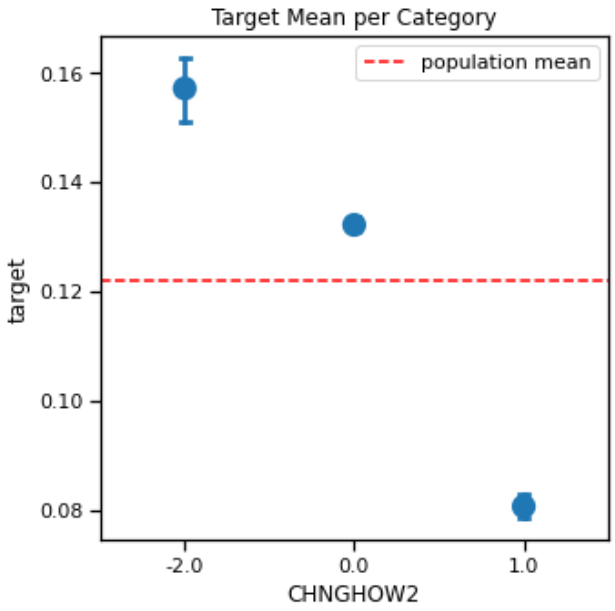

Distribution of CHNGHOW11
Count
CHNGHOW11
70000
60000
50000
40000
30000
20000
10000
0
−2
−1
0
1

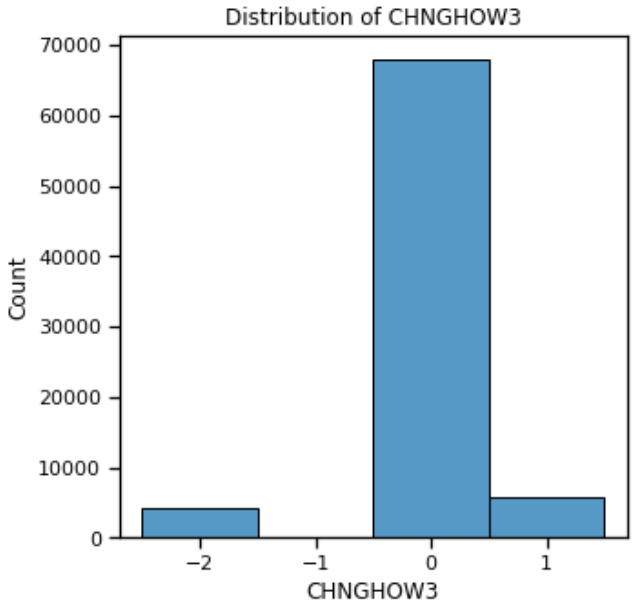

Target Mean per Category
population mean
target
CHNGHOW11
0.16
0.15
0.14
0.13
0.12
0.11
0.10
0.09
0.08
-2.0
0.0
1.0

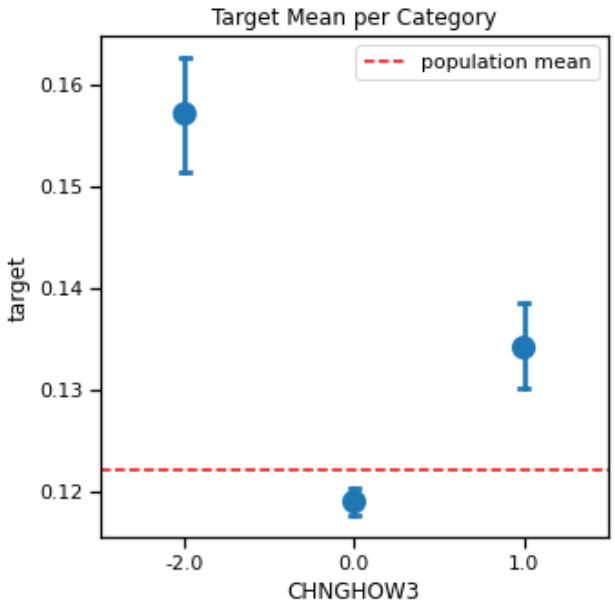

Distribution of CHNGHOW12
Count
CHNGHOW12
50000
40000
30000
20000
10000
0
−2
−1
0
1

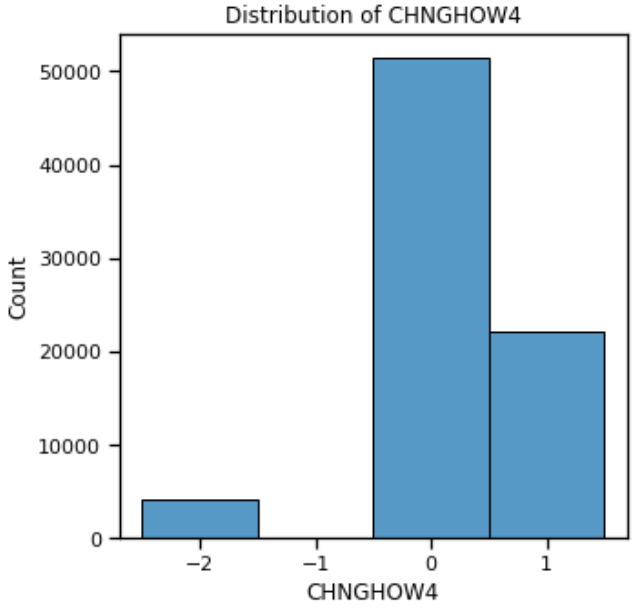

Target Mean per Category
population mean
target
CHNGHOW12
0.18
0.16
0.14
0.12
0.10
-2.0
0.0
1.0

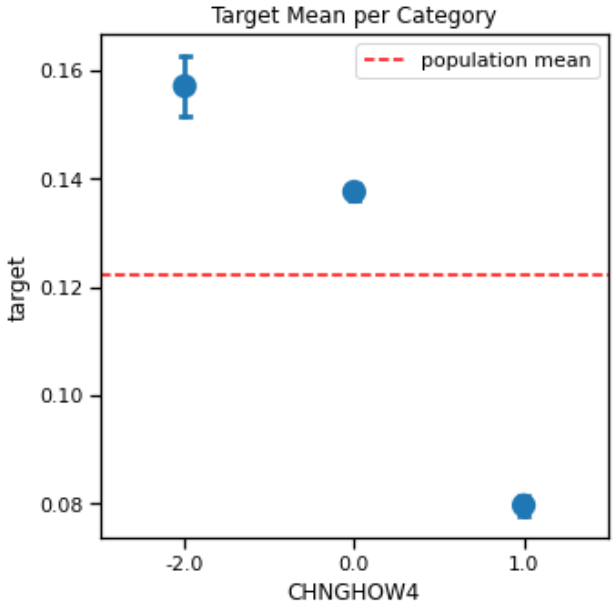

Distribution of CHNGHOW2
Count
CHNGHOW2
0
10000
20000
30000
40000
50000
−2
−1
0
1

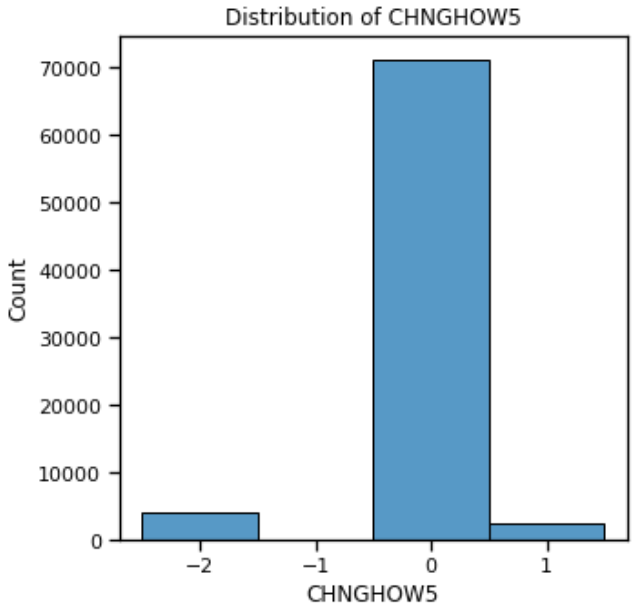

Target Mean per Category
population mean
target
CHNGHOW2
0.08
0.10
0.12
0.14
0.16
-2.0
0.0
1.0

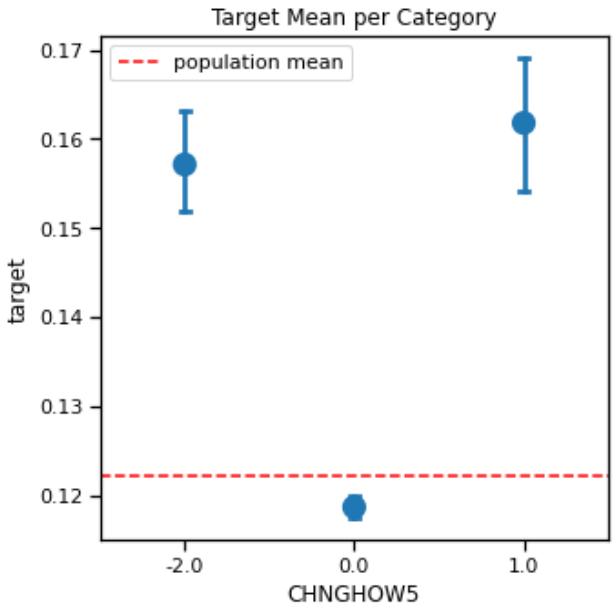

Distribution of CHNGHOW3
Count
CHNGHOW3
0
10000
20000
30000
40000
50000
60000
70000
−2
−1
0
1

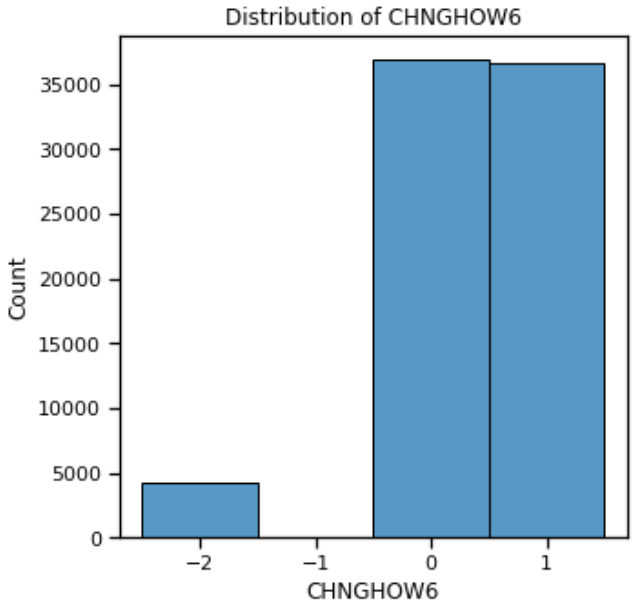

Target Mean per Category
population mean
target
CHNGHOW3
0.12
0.13
0.14
0.15
0.16
-2.0
0.0
1.0

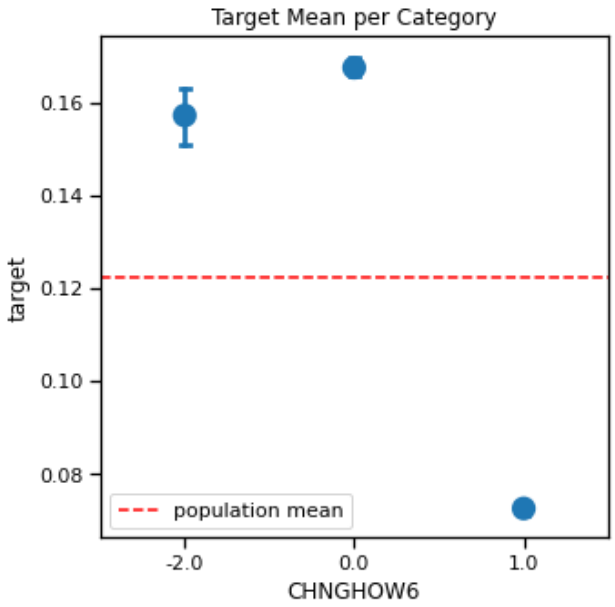

Distribution of CHNGHOW4
Count
CHNGHOW4
0
10000
20000
30000
40000
50000
−2
−1
0
1

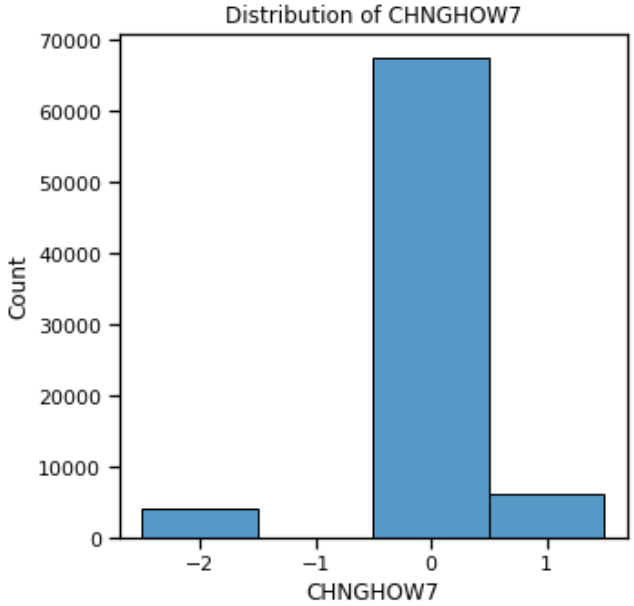

Target Mean per Category
population mean
target
CHNGHOW4
0.08
0.10
0.12
0.14
0.16
-2.0
0.0
1.0

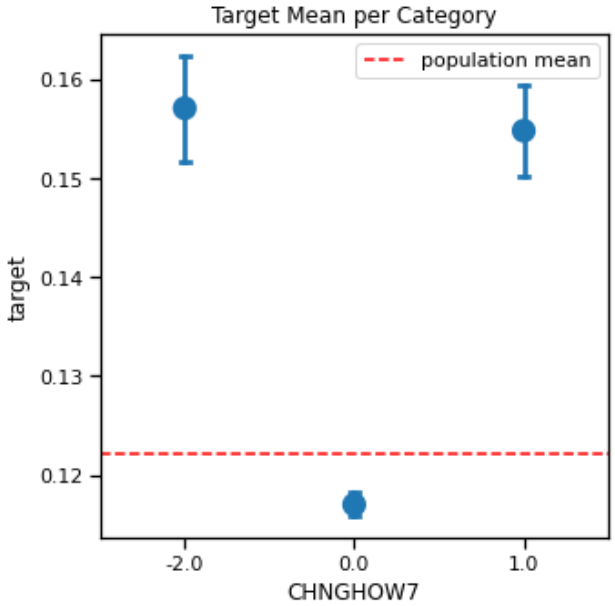

Distribution of CHNGHOW5
Count
CHNGHOW5
70000
60000
50000
40000
30000
20000
10000
0
−2
−1
0
1

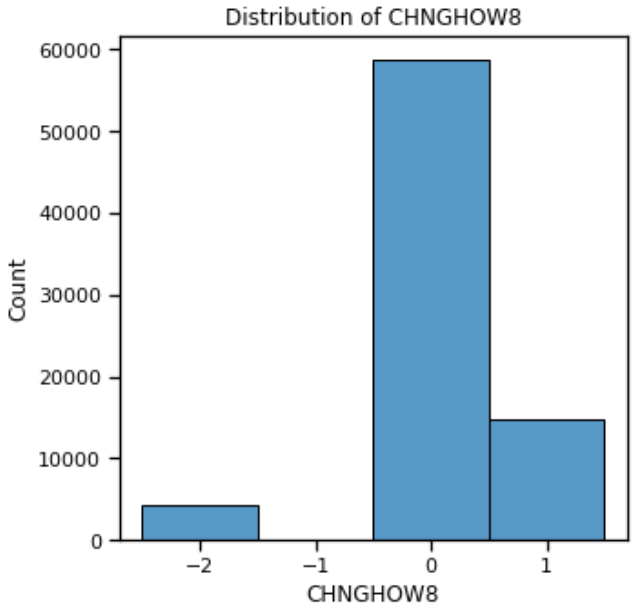

Target Mean per Category
target
CHNGHOW5
population mean
0.17
0.16
0.15
0.14
0.13
0.12
-2.0
0.0
1.0

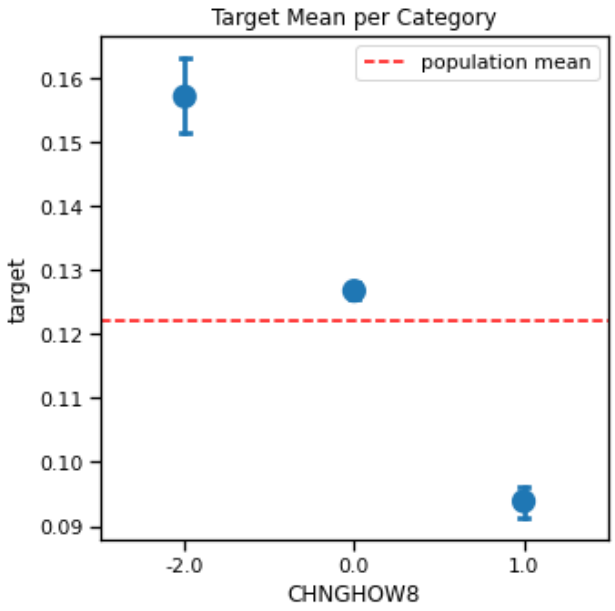

Distribution of CHNGHOW6
Count
CHNGHOW6
35000
30000
25000
20000
15000
10000
5000
0
−2
−1
0
1

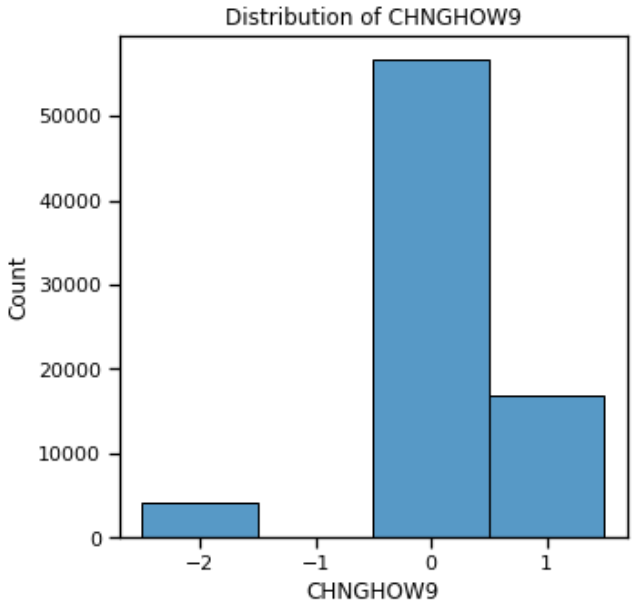

Target Mean per Category
target
CHNGHOW6
population mean
0.16
0.14
0.12
0.10
0.08
-2.0
0.0
1.0

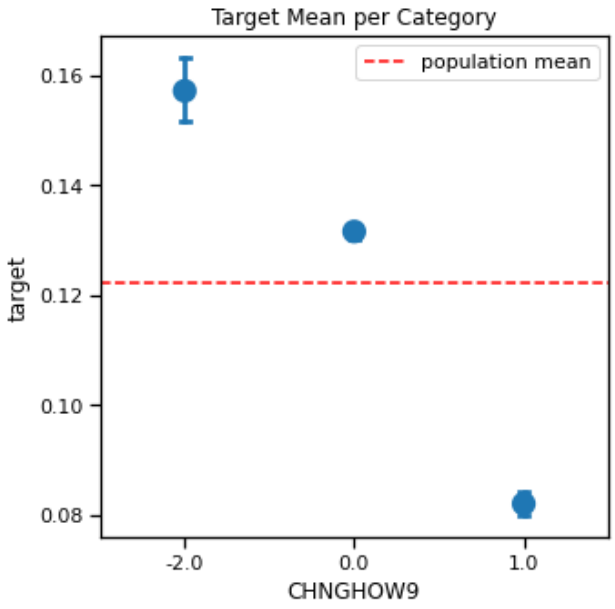

Distribution of CHNGHOW7
Count
CHNGHOW7
70000
60000
50000
40000
30000
20000
10000
0
−2
−1
0
1

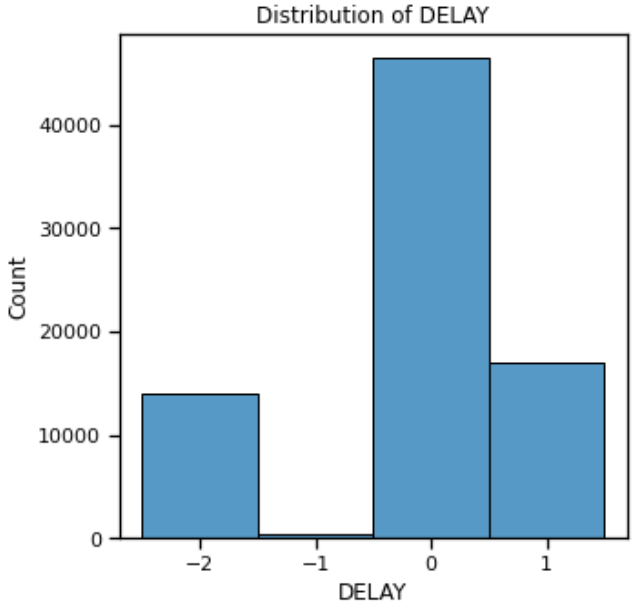

Target Mean per Category
target
CHNGHOW7
population mean
0.16
0.15
0.14
0.13
0.12
-2.0
0.0
1.0

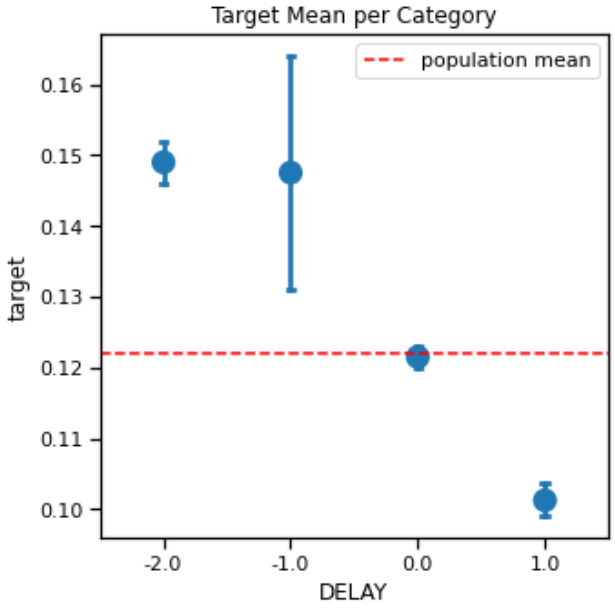

Distribution of CHNGHOW8
Count
CHNGHOW8
60000
50000
40000
30000
20000
10000
0
−2
−1
0
1

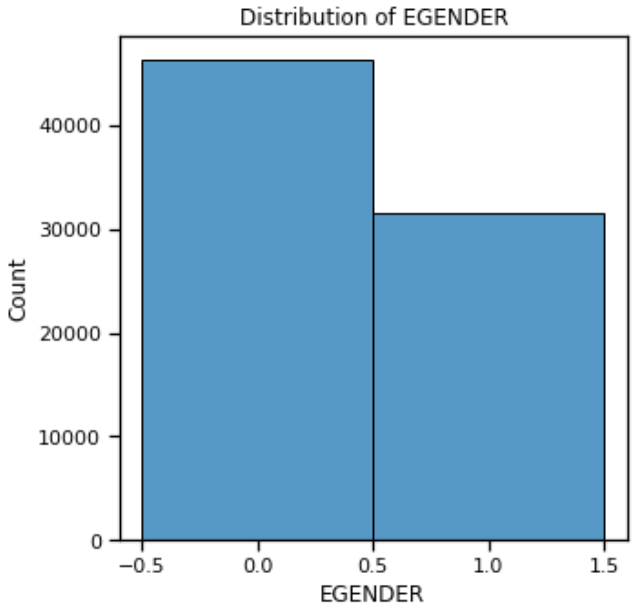

Target Mean per Category
population mean
target
CHNGHOW8
0.16
0.15
0.14
0.13
0.12
0.11
0.10
0.09
-2.0
0.0
1.0

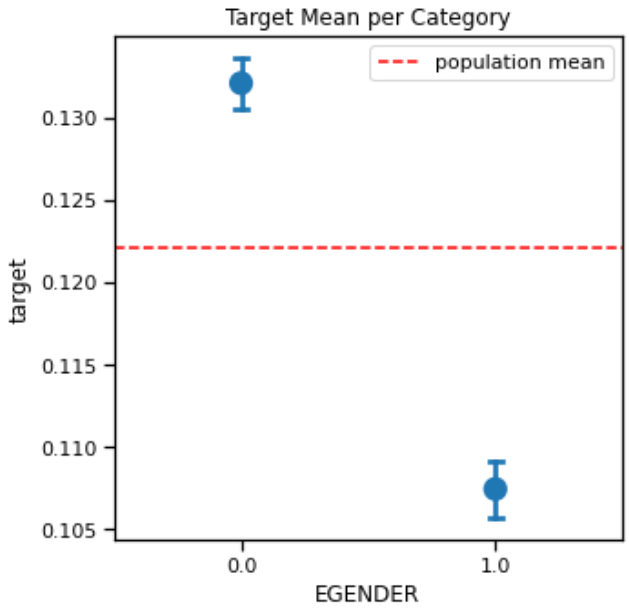

Distribution of CHNGHOW9
Count
CHNGHOW9
50000
40000
30000
20000
10000
0
−2
−1
0
1

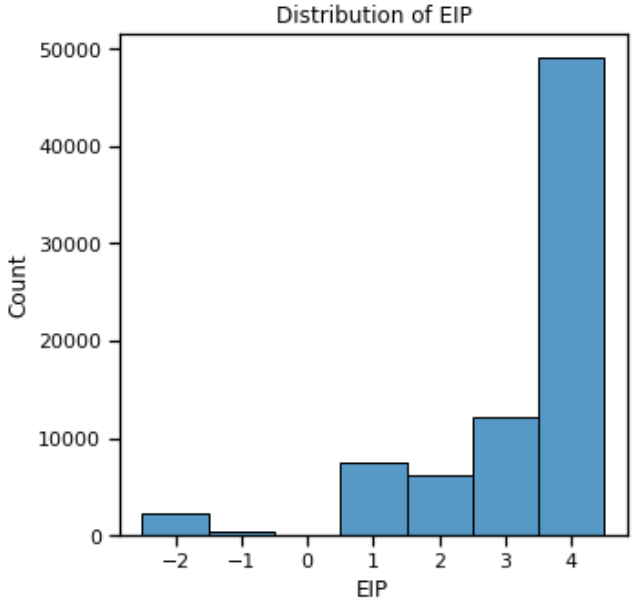

Target Mean per Category
population mean
target
CHNGHOW9
0.16
0.14
0.12
0.10
0.08
-2.0
0.0
1.0

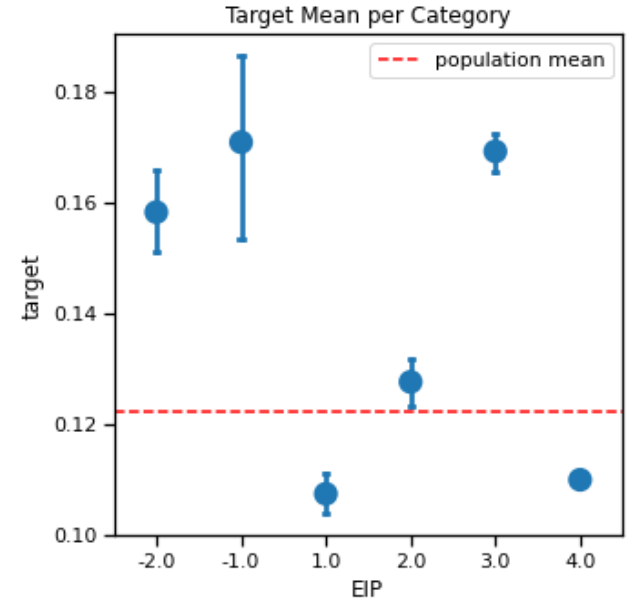

Distribution of DELAY
Count
DELAY
40000
30000
20000
10000
0
−2
−1
0
1

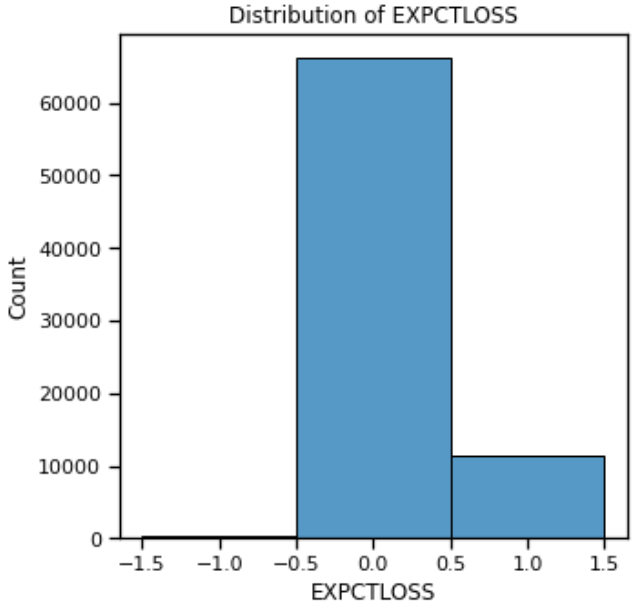

Target Mean per Category
population mean
target
DELAY
0.16
0.15
0.14
0.13
0.12
0.11
0.10
-2.0
-1.0
0.0
1.0

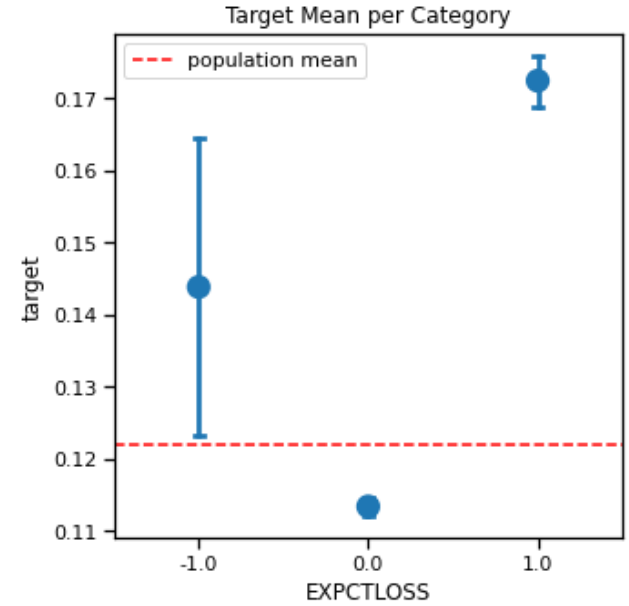

Distribution of EGENDER
Count
EGENDER
0
10000
20000
30000
40000
−0.5
0.0
0.5
1.0
1.5

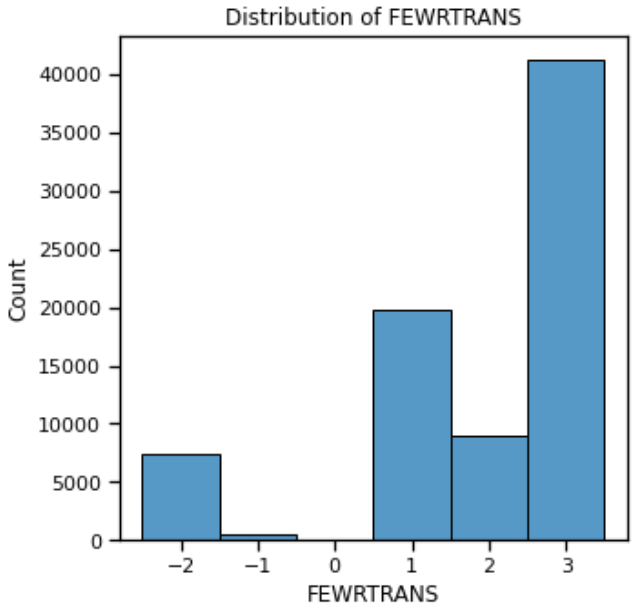

Target Mean per Category
population mean
target
EGENDER
0.105
0.110
0.115
0.120
0.125
0.130
0.0
1.0

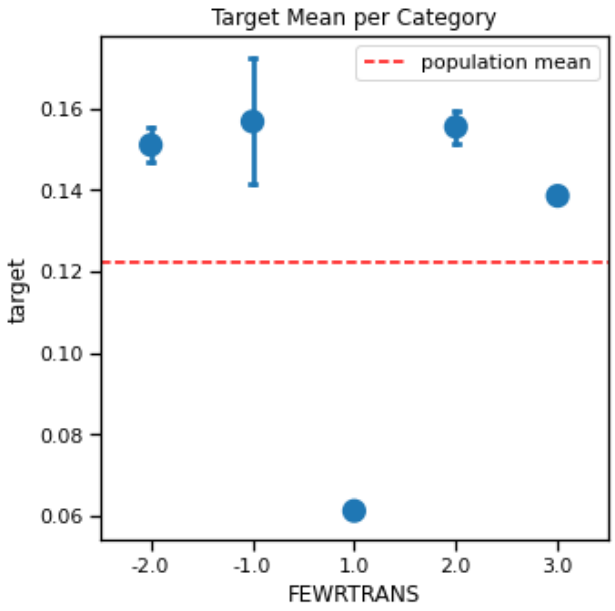

Distribution of EIP
Count
EIP
0
10000
20000
30000
40000
50000
−2
−1
0
1
2
3
4

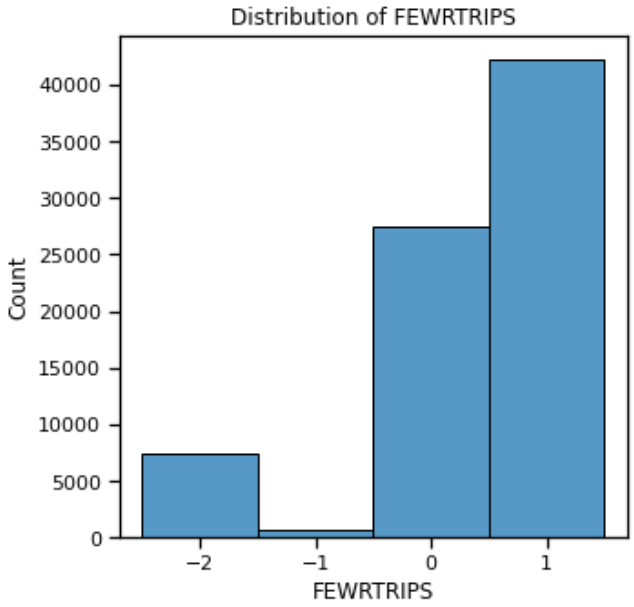

Target Mean per Category
population mean
target
EIP
0.10
0.12
0.14
0.16
0.18
-2.0
-1.0
1.0
2.0
3.0
4.0

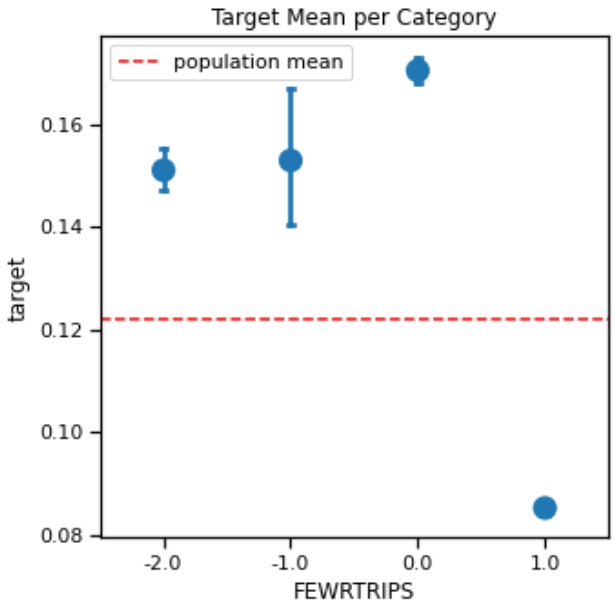

Distribution of EXPCTLOSS
Count
EXPCTLOSS
0
10000
20000
30000
40000
50000
60000
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

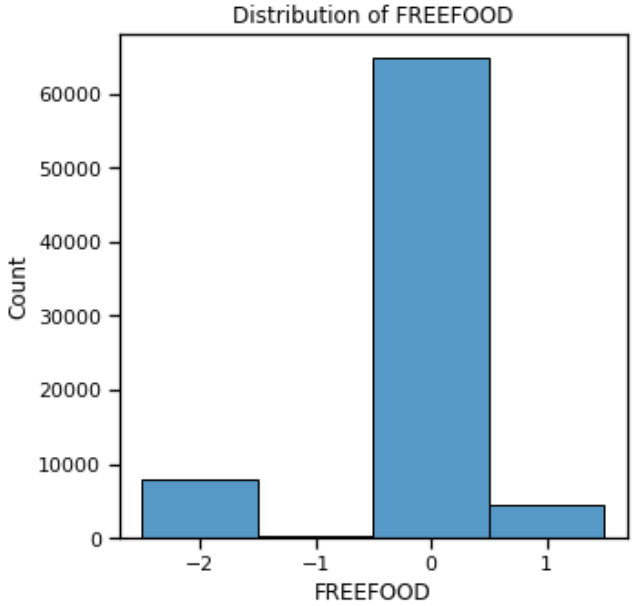

Target Mean per Category
population mean
target
EXPCTLOSS
0.11
0.12
0.13
0.14
0.15
0.16
0.17
-1.0
0.0
1.0

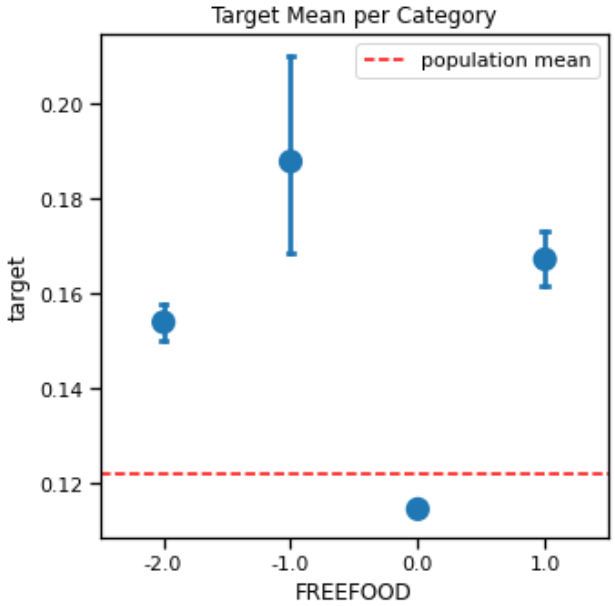

Distribution of FEWRTRANS
Count
FEWRTRANS

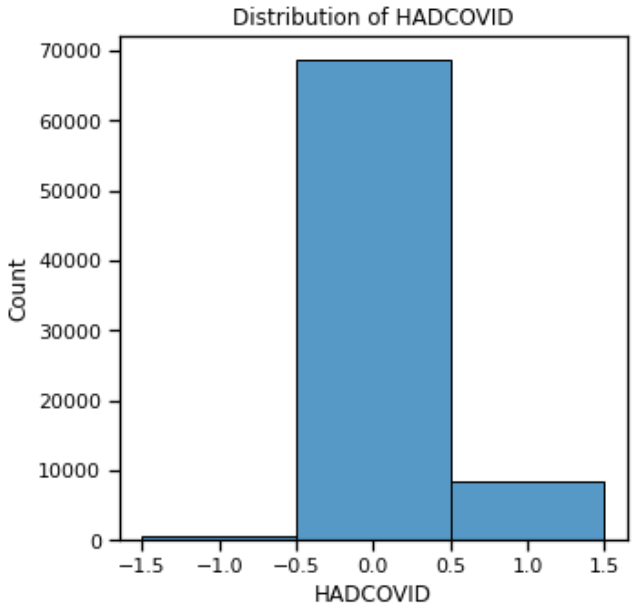

Target Mean per Category
population mean
target
FEWRTRANS

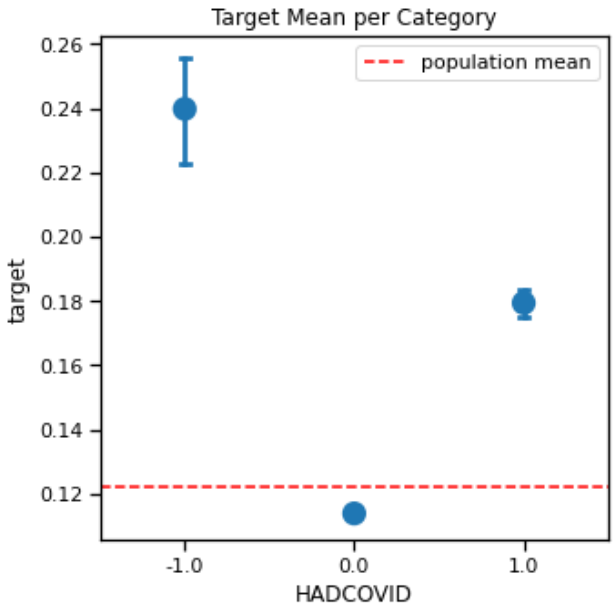

Distribution of FEWRTRIPS
Count
FEWRTRIPS

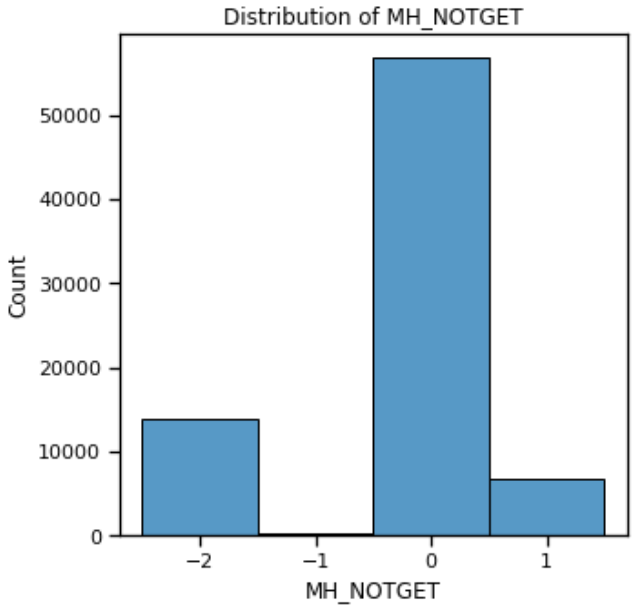

Target Mean per Category
population mean
target
FEWRTRIPS

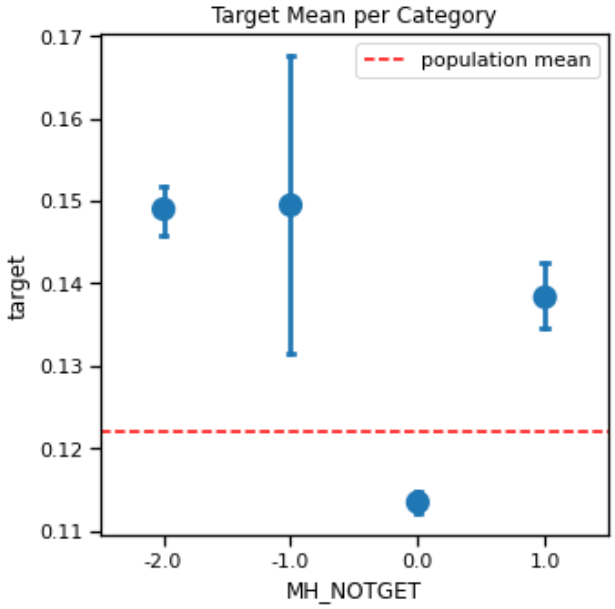

Distribution of FREEFOOD
Count
FREEFOOD

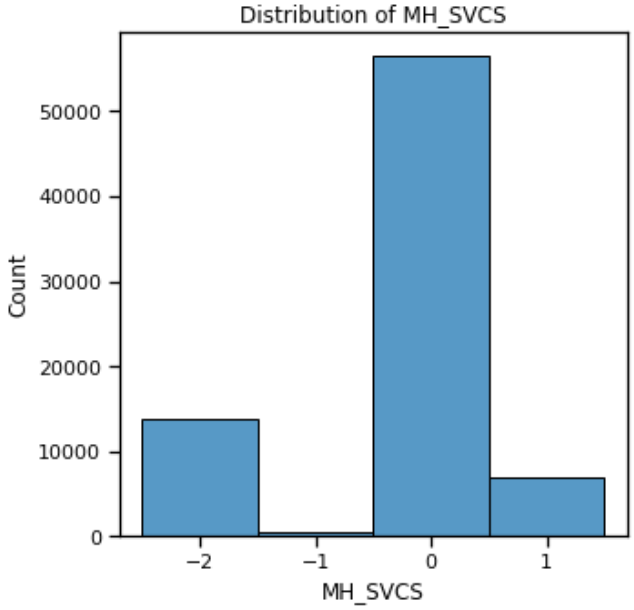

Target Mean per Category
population mean
target
FREEFOOD

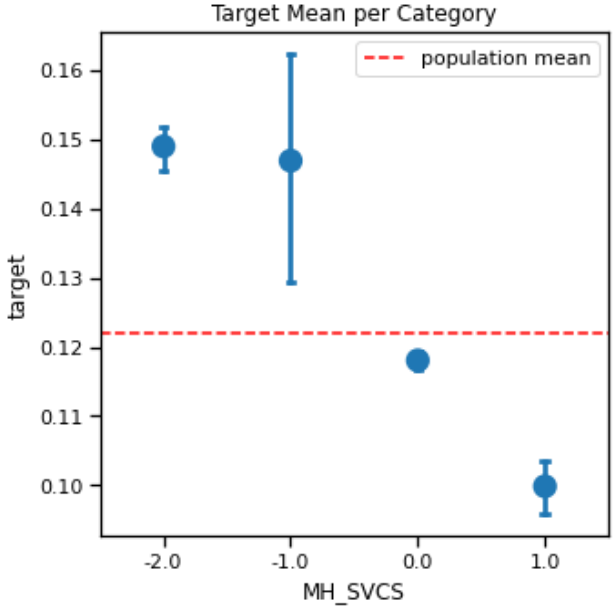

Distribution of HADCOVID
Count
HADCOVID

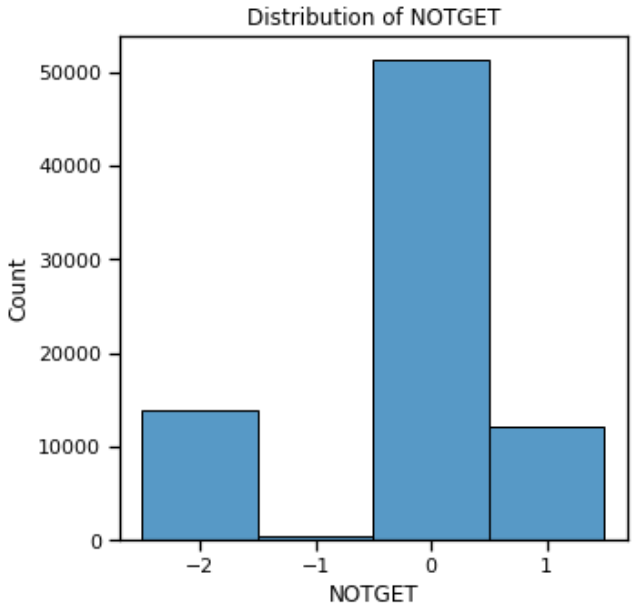

Target Mean per Category
population mean
target
HADCOVID

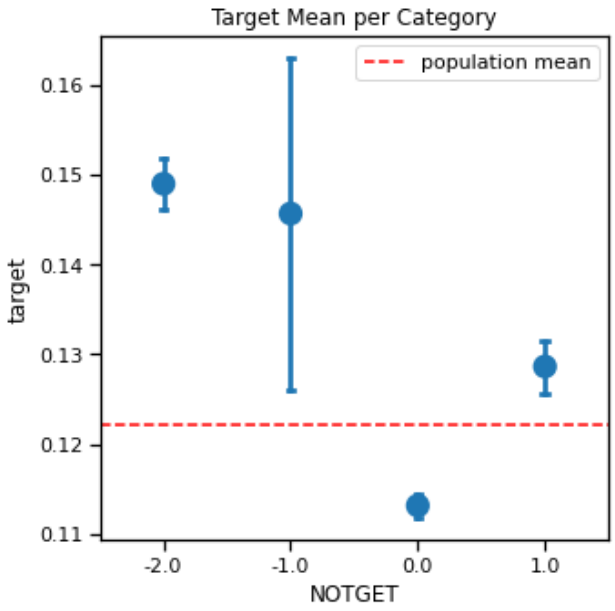

Distribution of MH_NOTGET
Count
MH_NOTGET

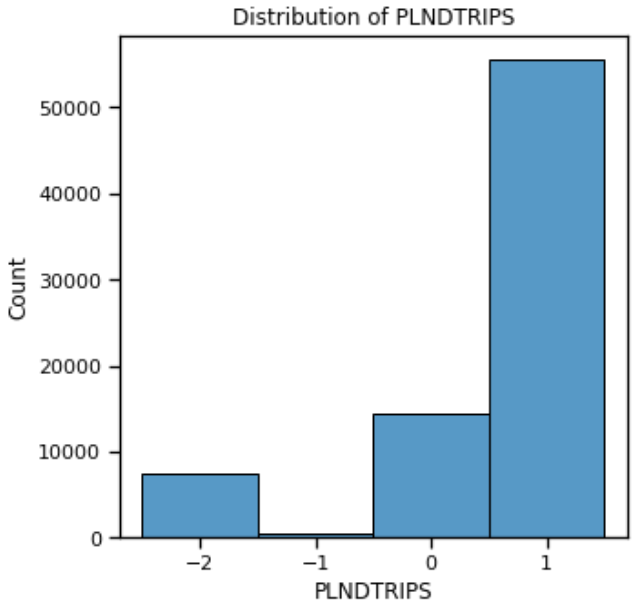

Target Mean per Category
population mean
target
MH_NOTGET

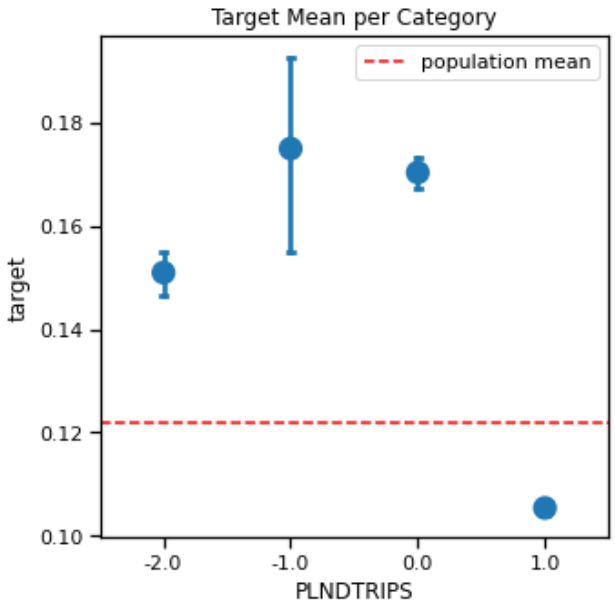

Distribution of MH_SVCS
Count
MH_SVCS

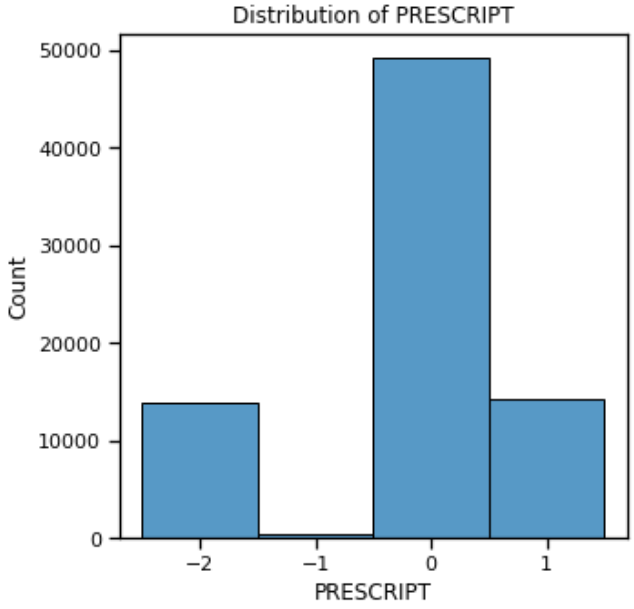

Target Mean per Category
population mean
target
MH_SVCS

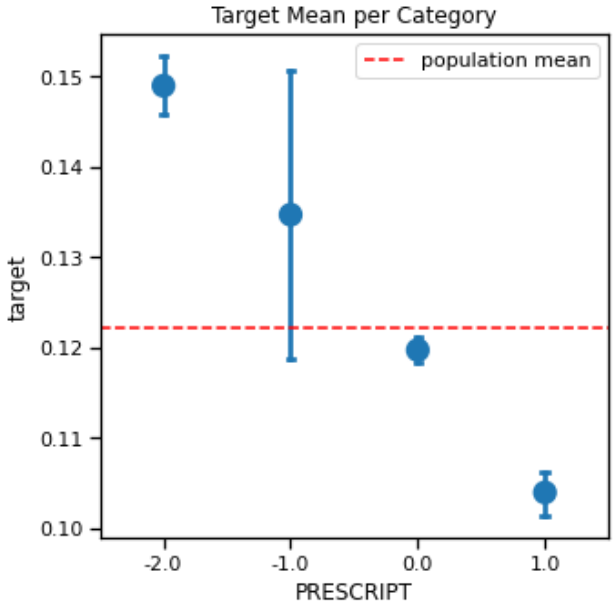

Distribution of NOTGET
Count
NOTGET

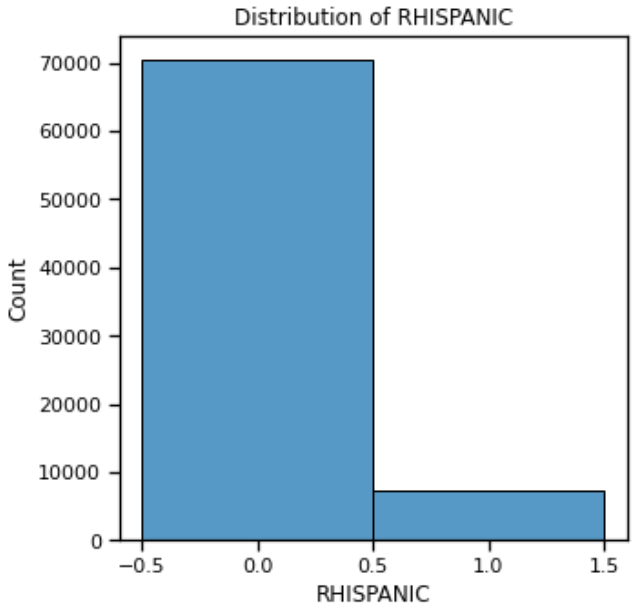

Target Mean per Category
population mean
target
NOTGET

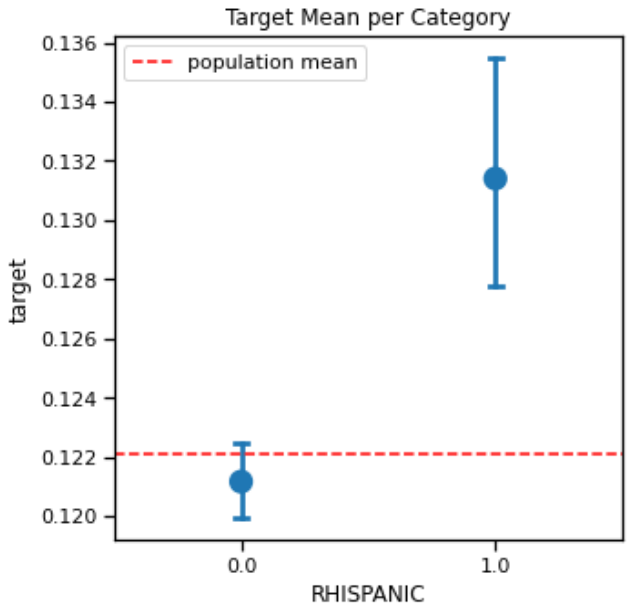

Distribution of PLNDTRIPS
Count
PLNDTRIPS

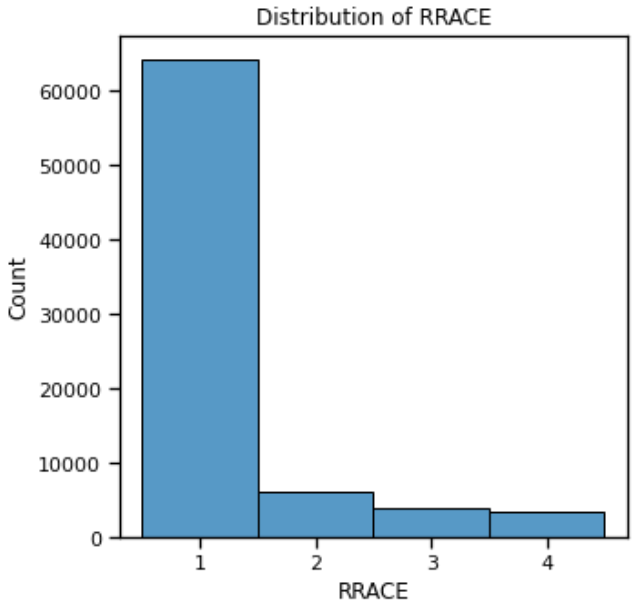

Target Mean per Category
population mean
target
PLNDTRIPS

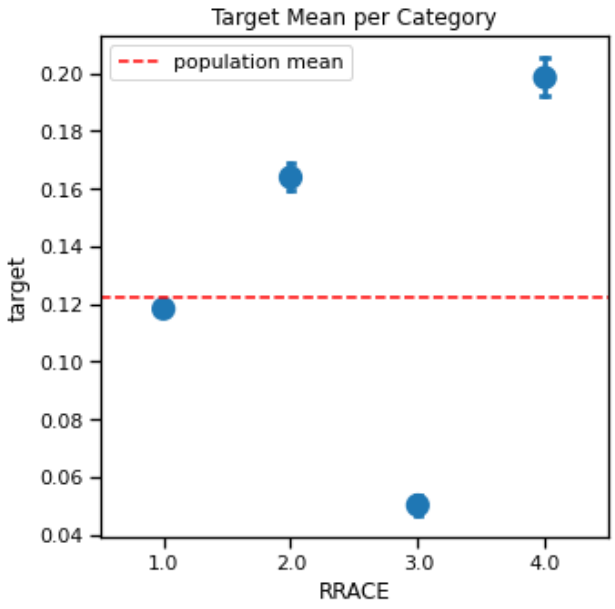

Distribution of PRESCRIPT
Count
PRESCRIPT

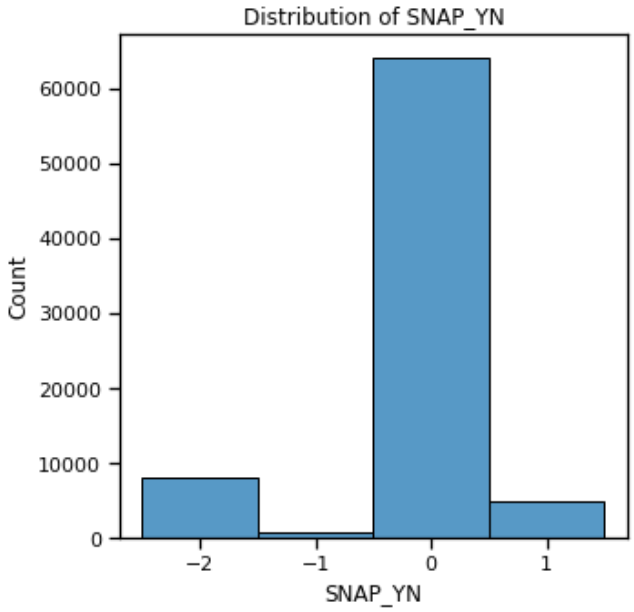

Target Mean per Category
population mean
target
PRESCRIPT

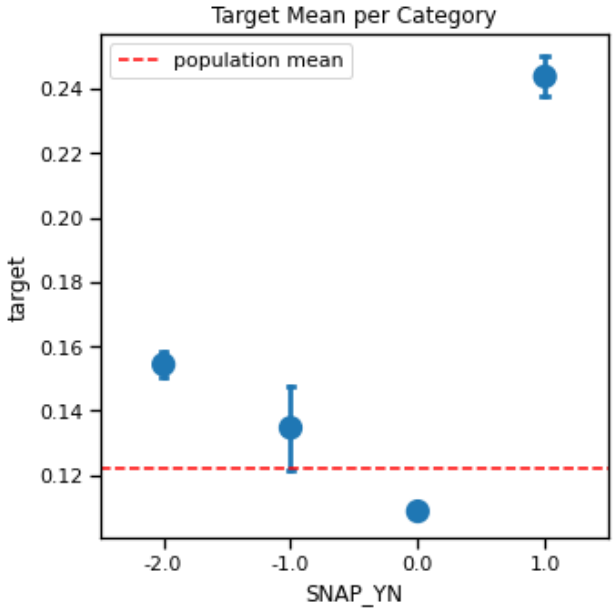

Distribution of RHISPANIC
Count
RHISPANIC

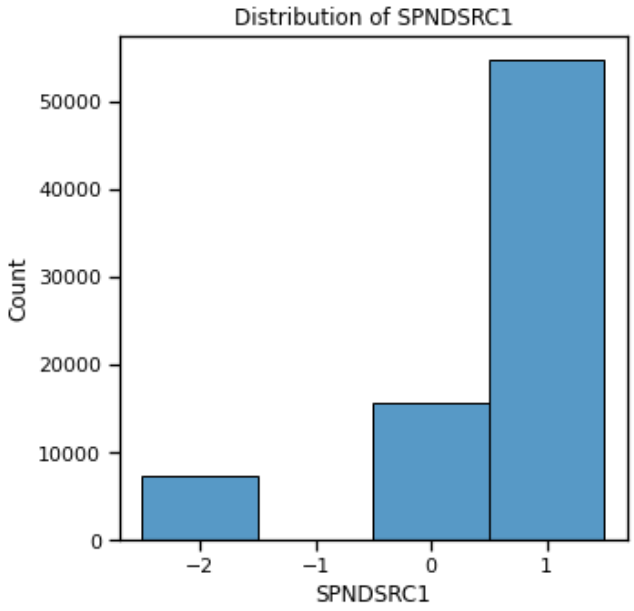

Target Mean per Category
population mean
target
RHISPANIC

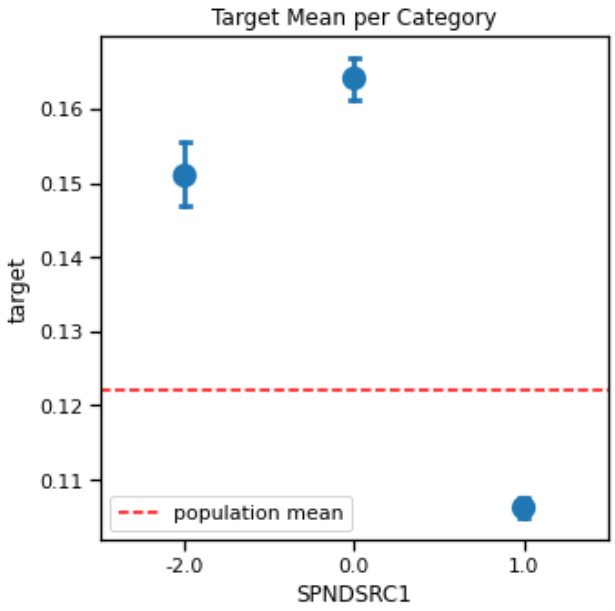

Distribution of RRACE
Count
RRACE

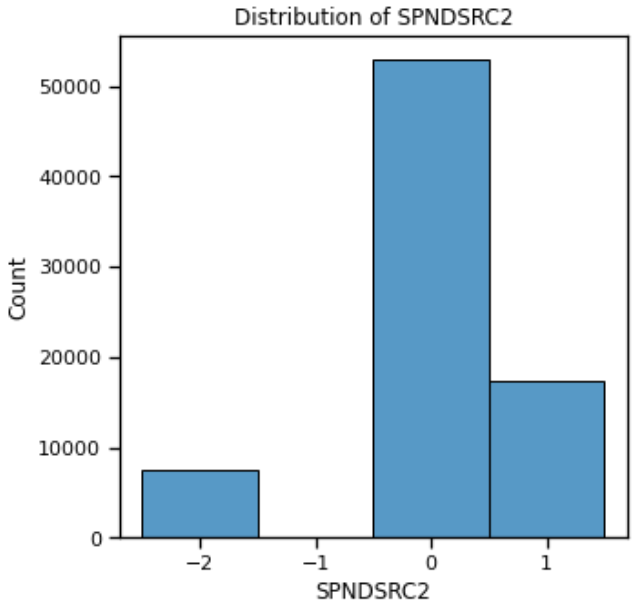

Target Mean per Category
population mean
target
RRACE

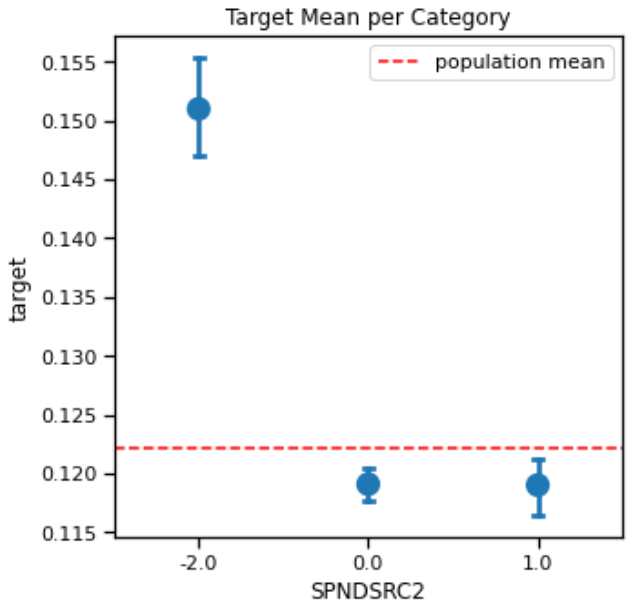

Distribution of SNAP_YN
Count
SNAP_YN

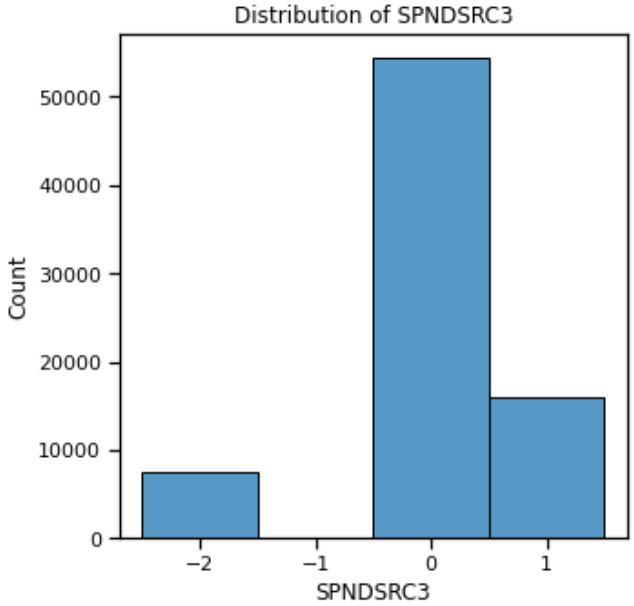

Target Mean per Category
population mean
target
SNAP_YN

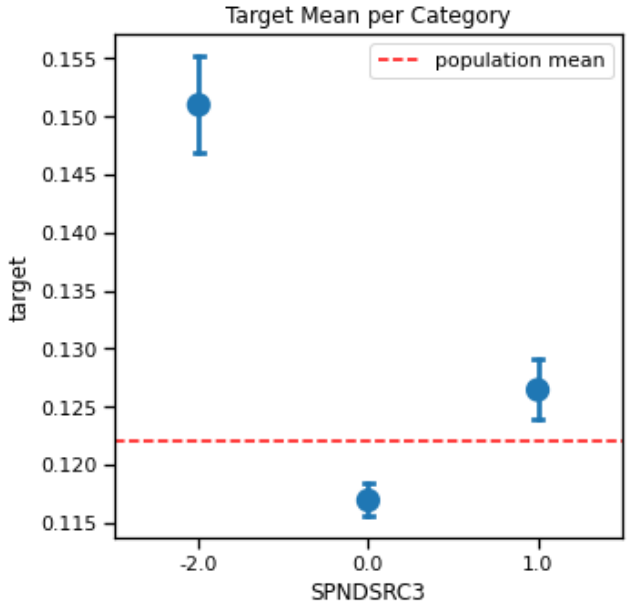

Distribution of SPNDSRC1
Count
SPNDSRC1
50000
40000
30000
20000
10000
0
−2
−1
0
1

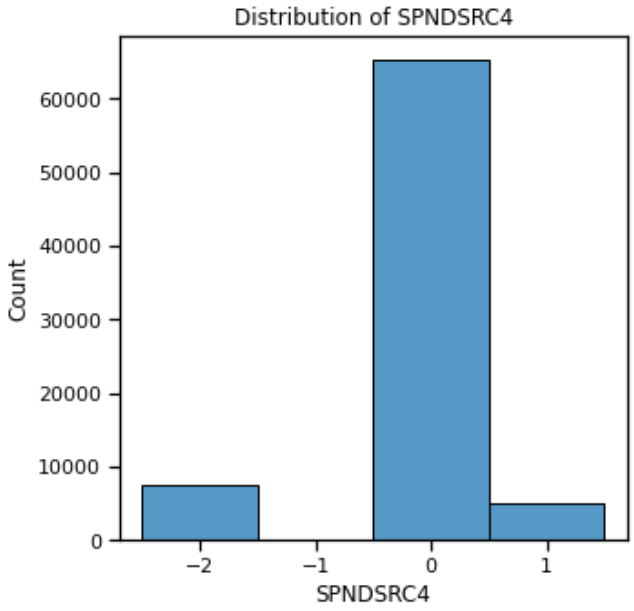

Target Mean per Category
target
SPNDSRC1
population mean
0.16
0.15
0.14
0.13
0.12
0.11
-2.0
0.0
1.0

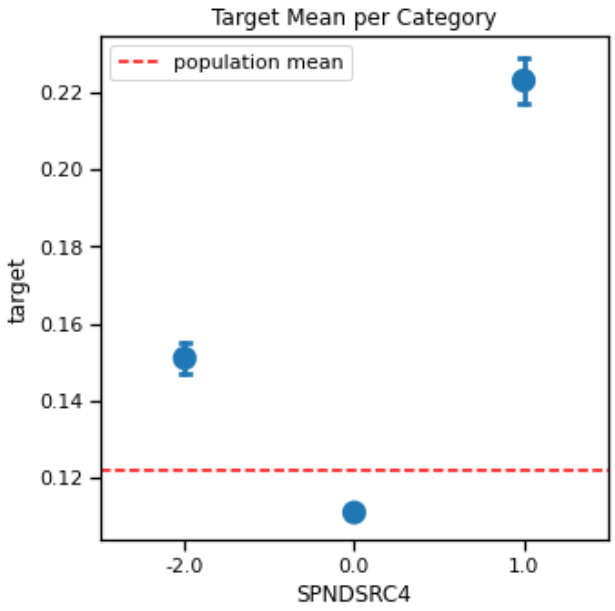

Distribution of SPNDSRC2
Count
SPNDSRC2
50000
40000
30000
20000
10000
0
−2
−1
0
1

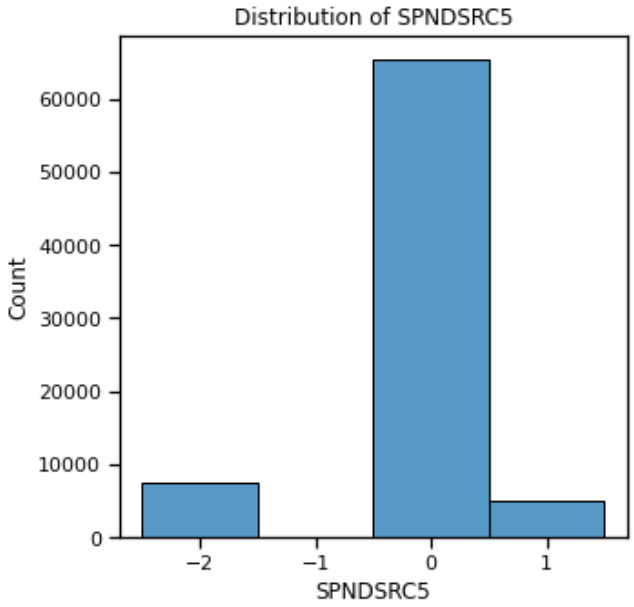

Target Mean per Category
target
SPNDSRC2
population mean
0.155
0.150
0.145
0.140
0.135
0.130
0.125
0.120
0.115
-2.0
0.0
1.0

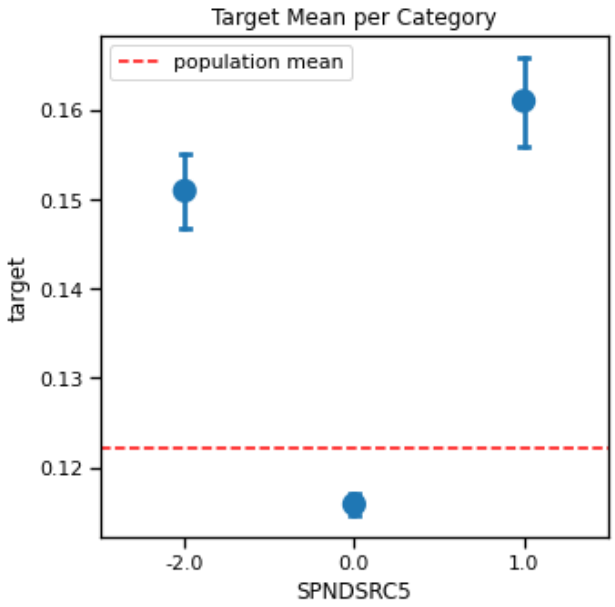

Distribution of SPNDSRC3
Count
SPNDSRC3
50000
40000
30000
20000
10000
0
−2
−1
0
1

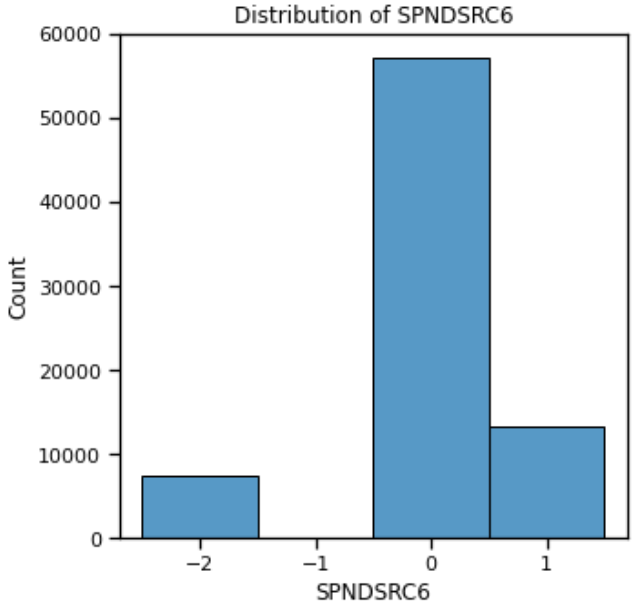

Target Mean per Category
target
SPNDSRC3
population mean
0.155
0.150
0.145
0.140
0.135
0.130
0.125
0.120
0.115
-2.0
0.0
1.0

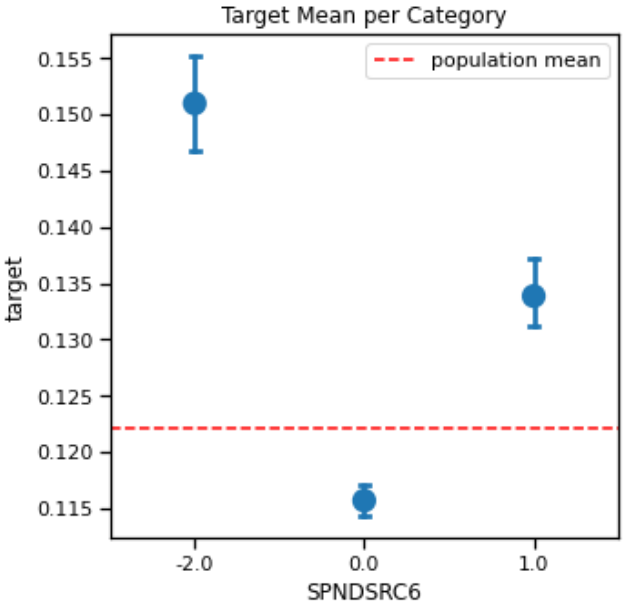

Distribution of SPNDSRC4
Count
SPNDSRC4
0
10000
20000
30000
40000
50000
60000
−2
−1
0
1

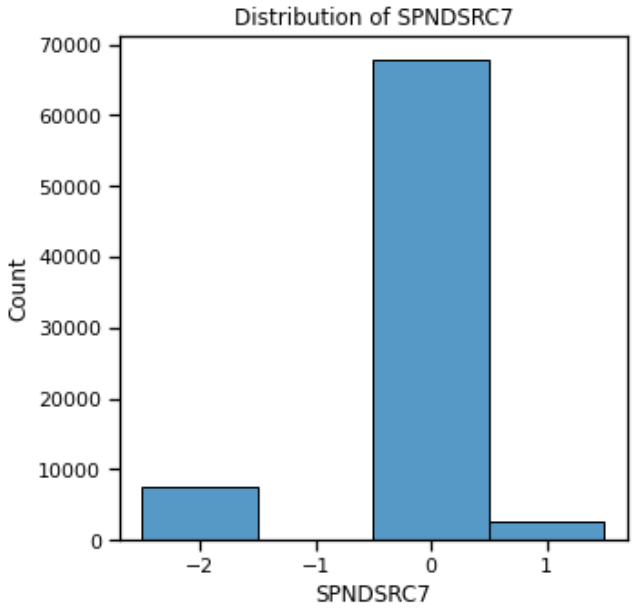

Target Mean per Category
population mean
target
SPNDSRC4
0.12
0.14
0.16
0.18
0.20
0.22
-2.0
0.0
1.0

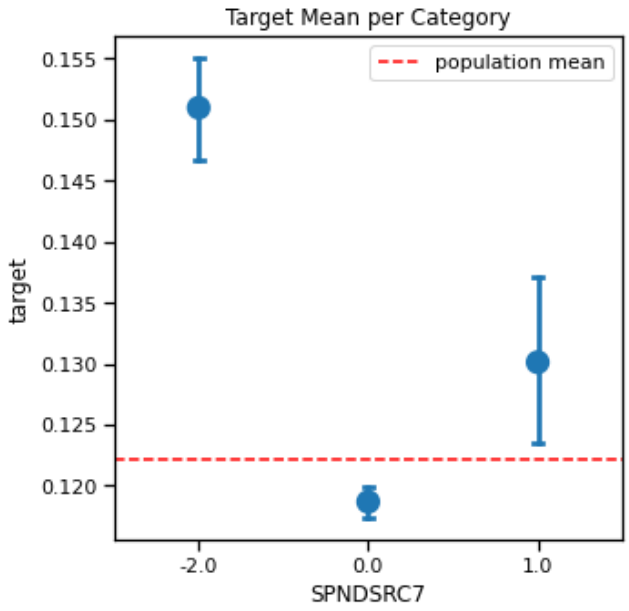

Distribution of SPNDSRC5
Count
SPNDSRC5
0
10000
20000
30000
40000
50000
60000
−2
−1
0
1

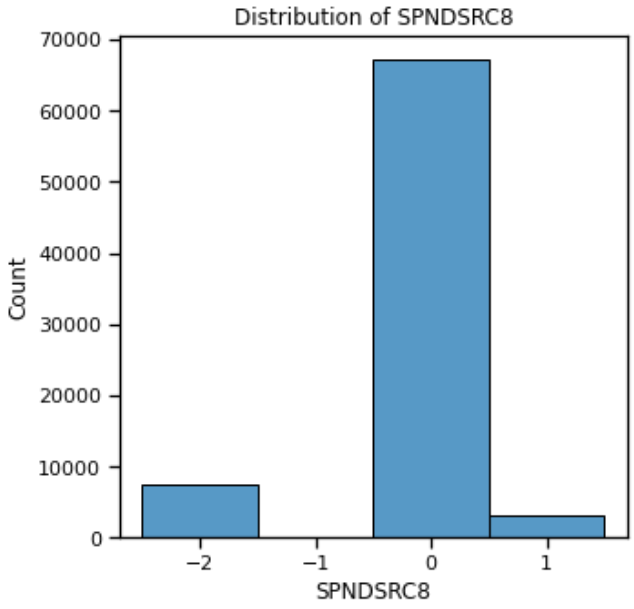

Target Mean per Category
population mean
target
SPNDSRC5
0.12
0.13
0.14
0.15
0.16
-2.0
0.0
1.0

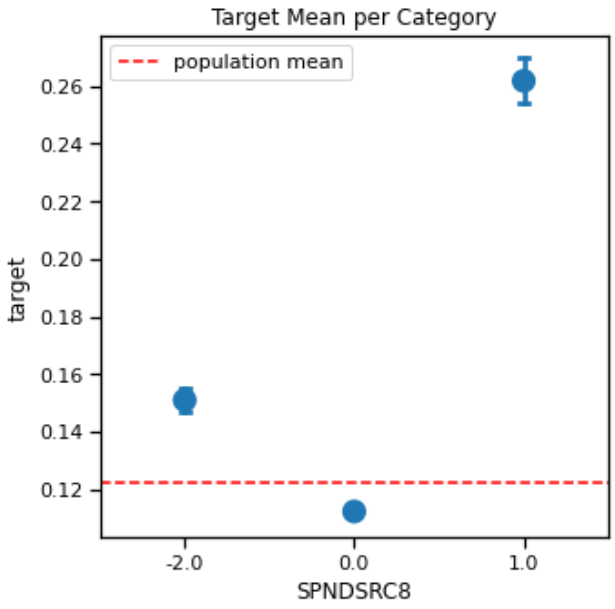

Distribution of SPNDSRC6
Count
SPNDSRC6
0
10000
20000
30000
40000
50000
60000
−2
−1
0
1

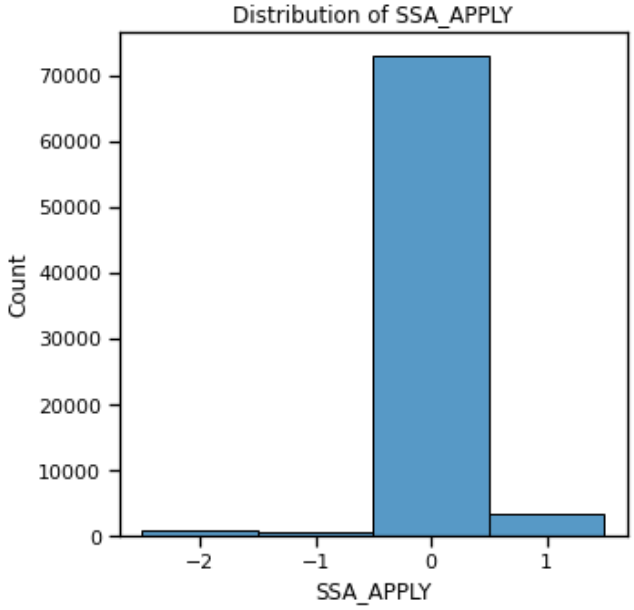

Target Mean per Category
population mean
target
SPNDSRC6
0.115
0.120
0.125
0.130
0.135
0.140
0.145
0.150
0.155
-2.0
0.0
1.0

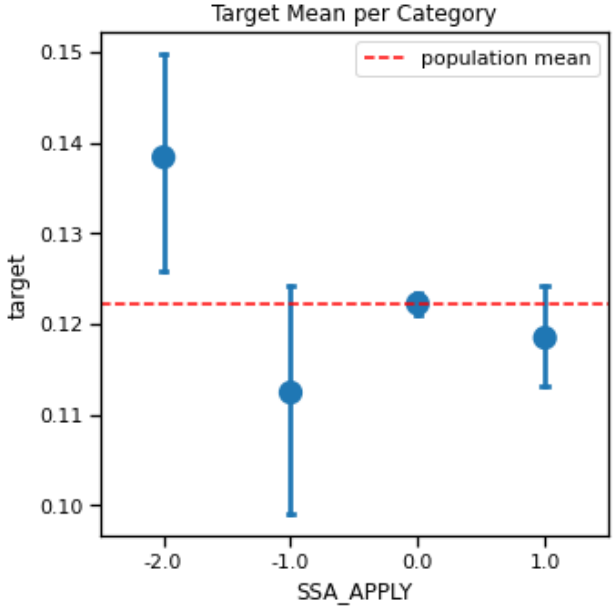

Distribution of SPNDSRC7
Count
SPNDSRC7

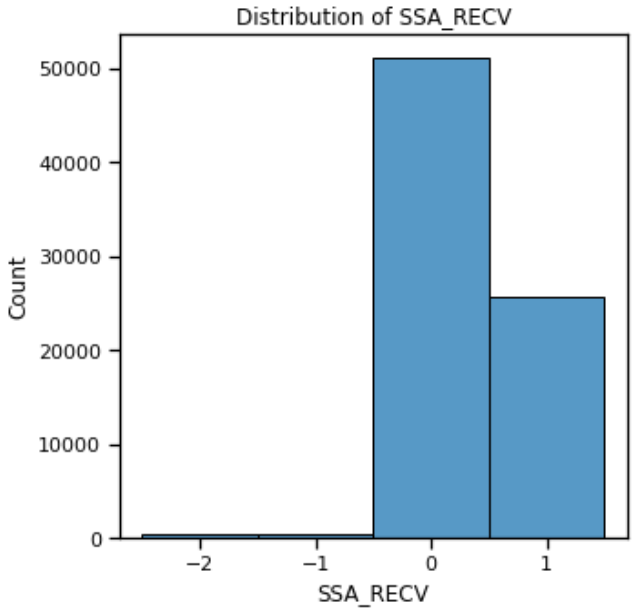

Target Mean per Category
population mean
target
SPNDSRC7

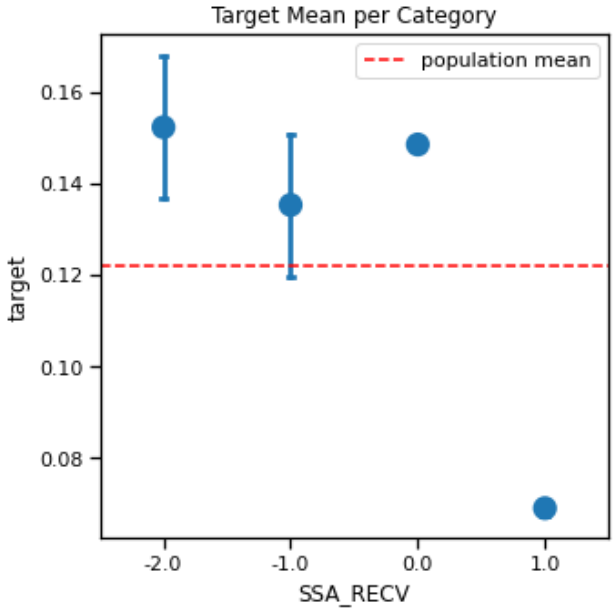

Distribution of SPNDSRC8
Count
SPNDSRC8

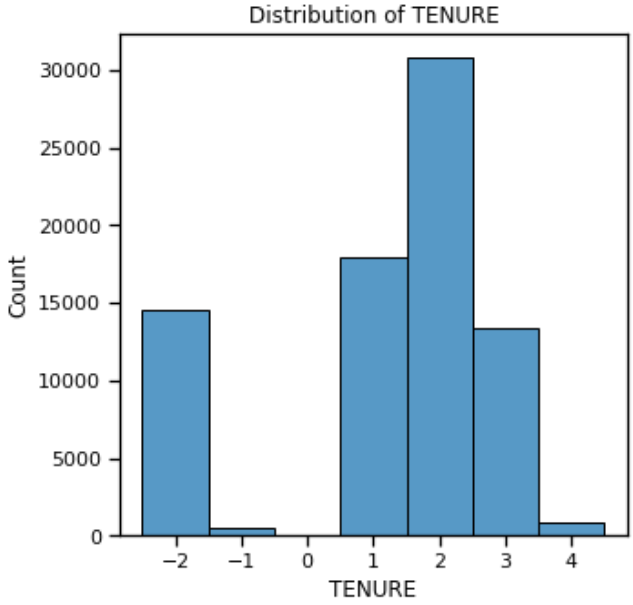

Target Mean per Category
population mean
target
SPNDSRC8

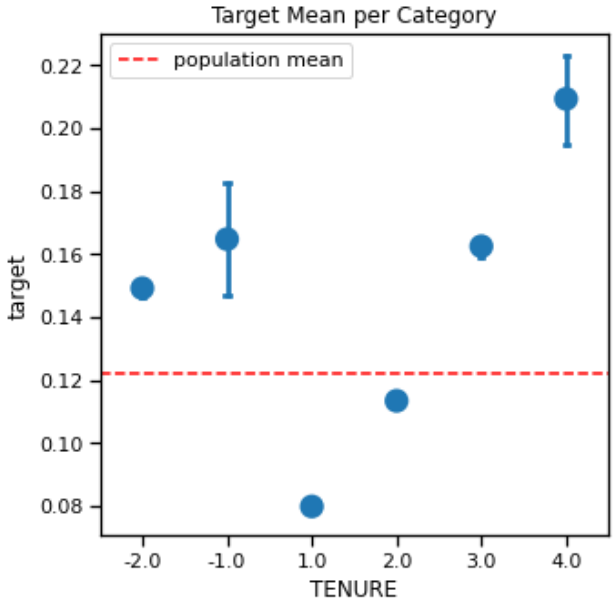

Distribution of SSA_APPLY
Count
SSA_APPLY

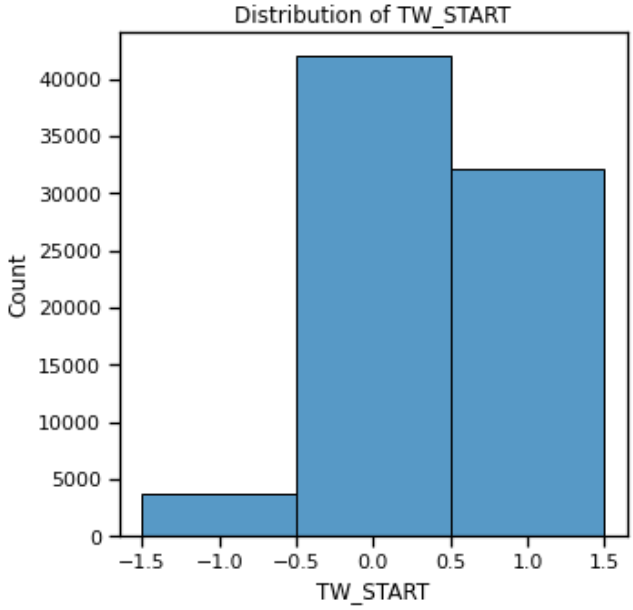

Target Mean per Category
population mean
target
SSA_APPLY

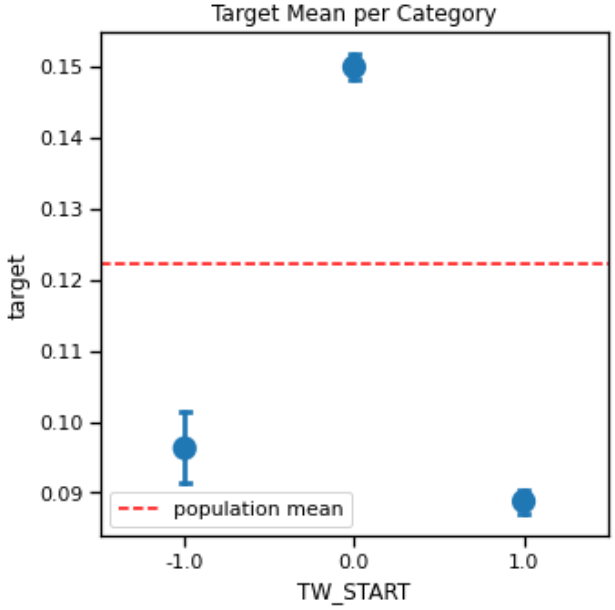

Distribution of SSA_RECV
Count
SSA_RECV

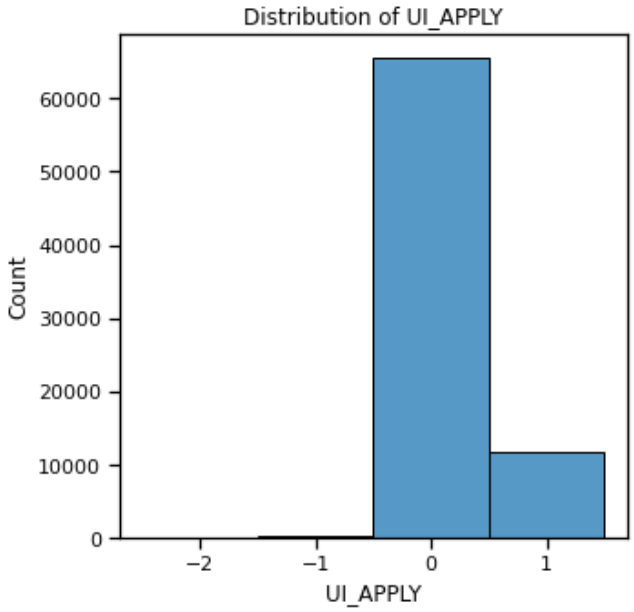

Target Mean per Category
population mean
target
SSA_RECV

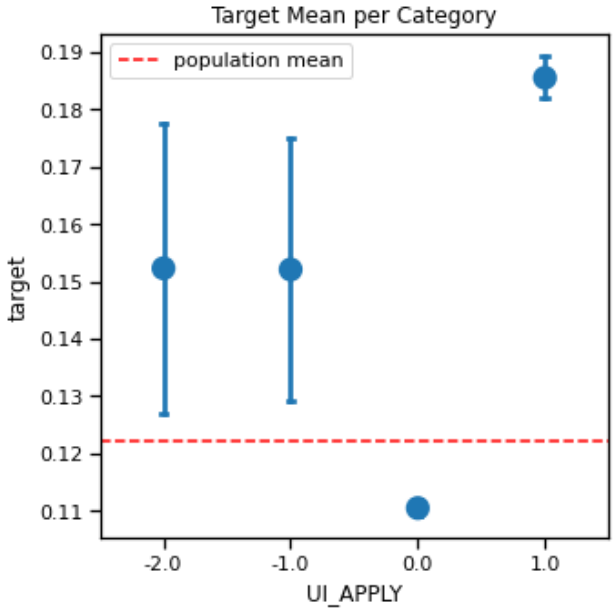

Distribution of TENURE
Count
TENURE

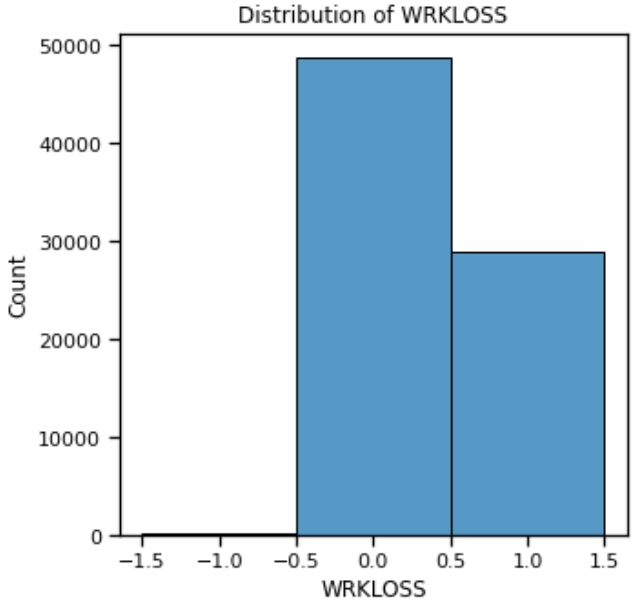

Target Mean per Category
population mean
target
TENURE

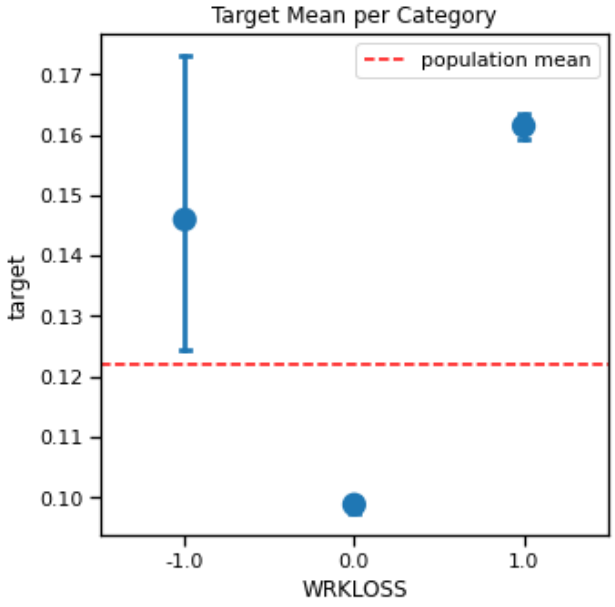

Distribution of TW_START
Count
TW_START

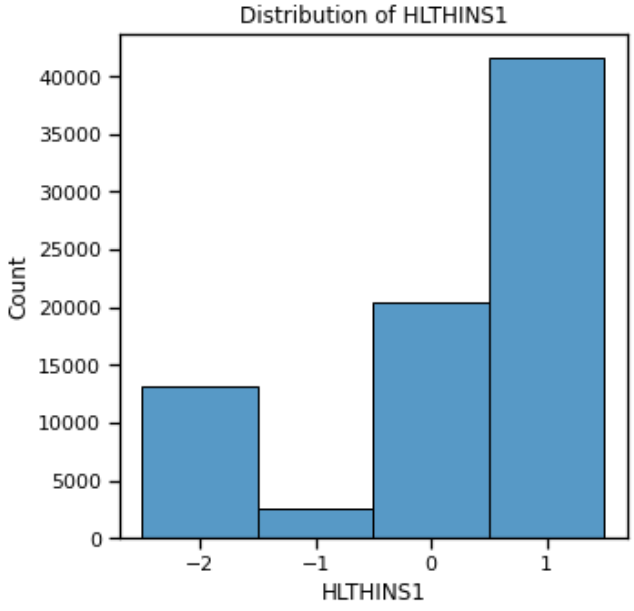

Target Mean per Category
population mean
target
TW_START

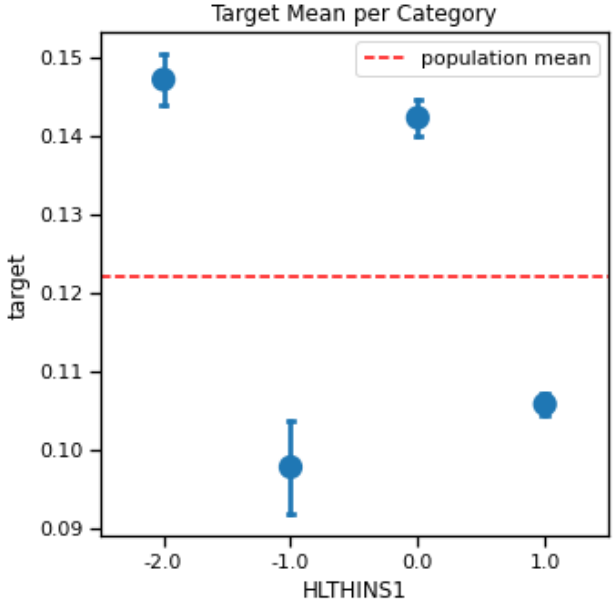

Distribution of UI_APPLY
Count
UI_APPLY

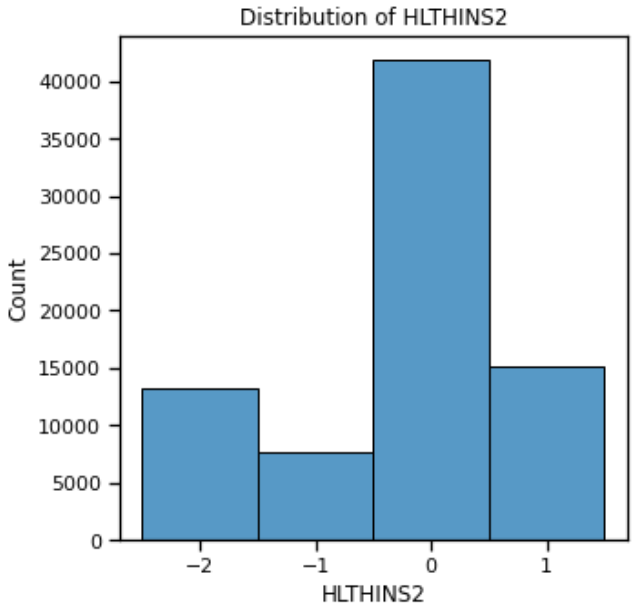

Target Mean per Category
population mean
target
UI_APPLY

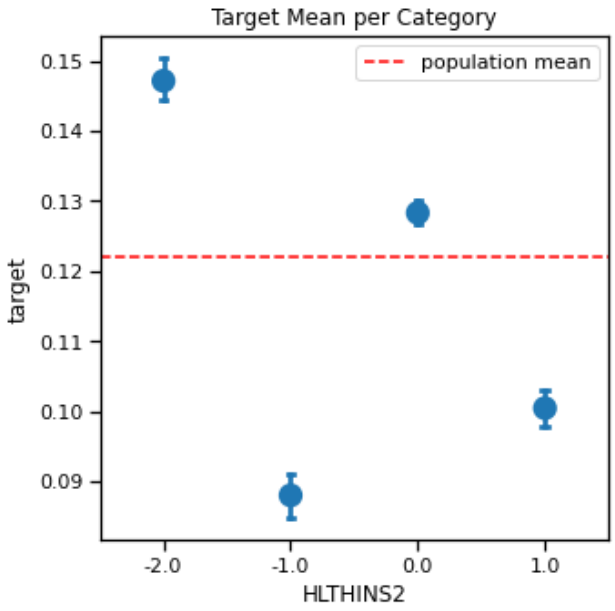

Distribution of WRKLOSS
Count
WRKLOSS

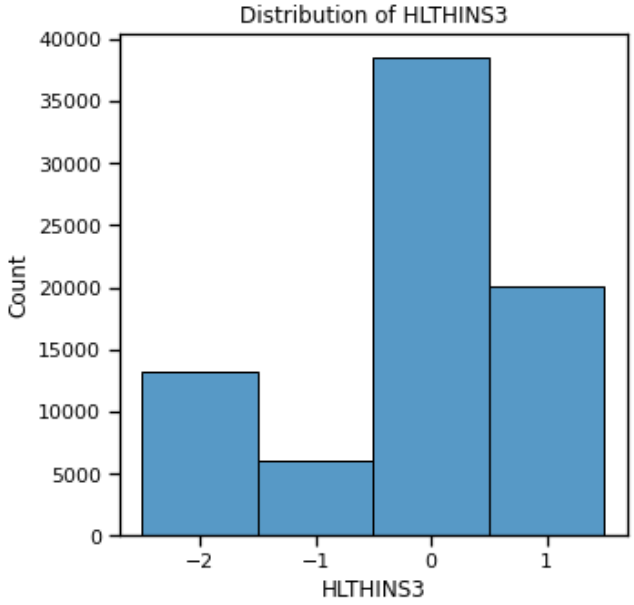

Target Mean per Category
population mean
target
WRKLOSS

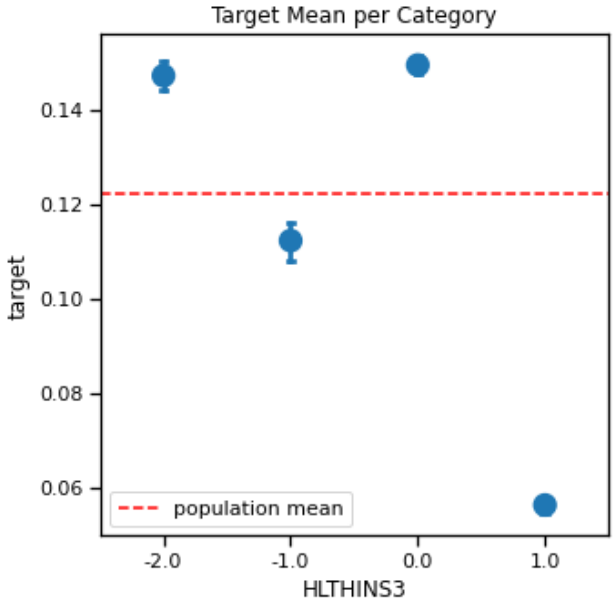

Distribution of HLTHINS1
Count
HLTHINS1

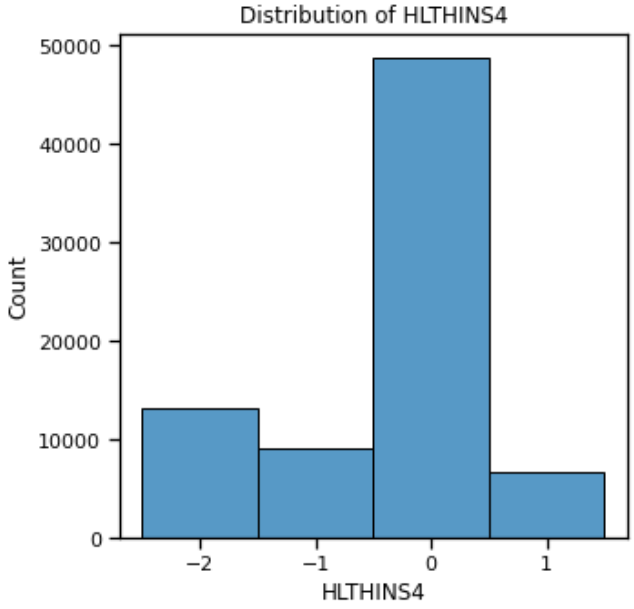

Target Mean per Category
population mean
target
HLTHINS1

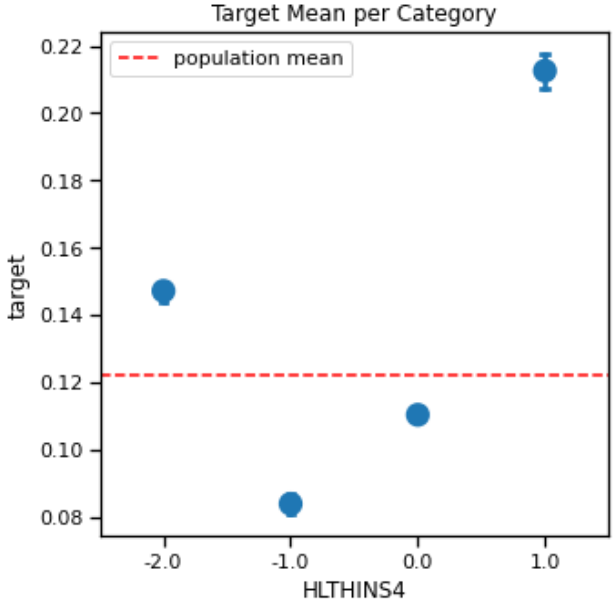

Distribution of HLTHINS2
Count
HLTHINS2

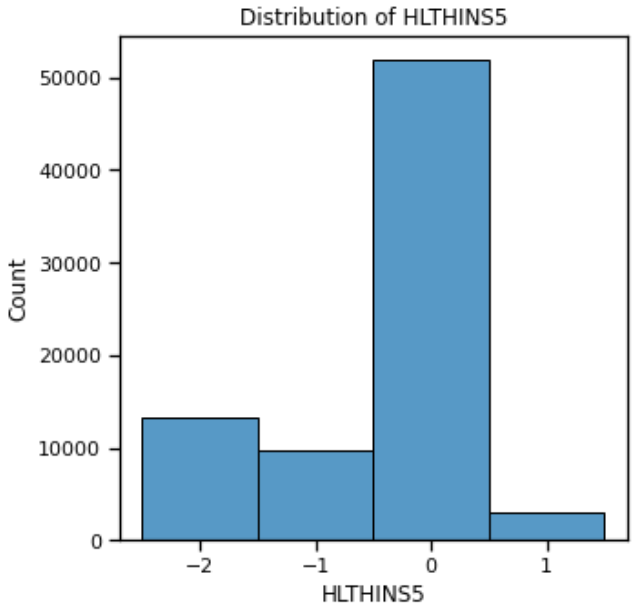

Target Mean per Category
population mean
target
HLTHINS2

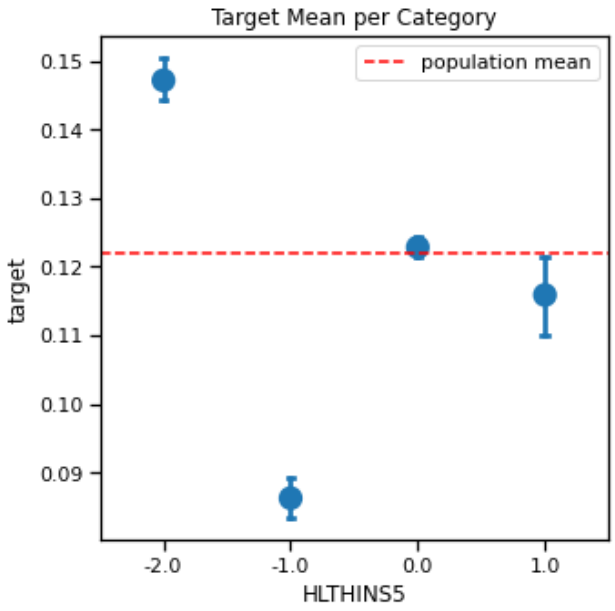

Distribution of HLTHINS3
Count
HLTHINS3

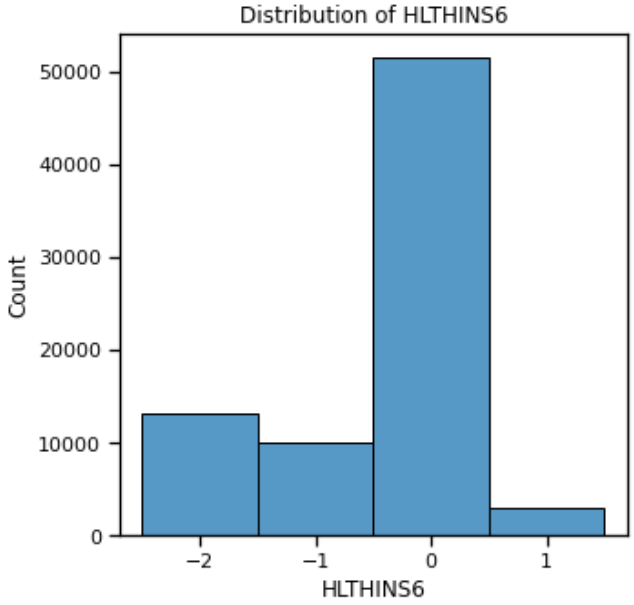

Target Mean per Category
population mean
target
HLTHINS3

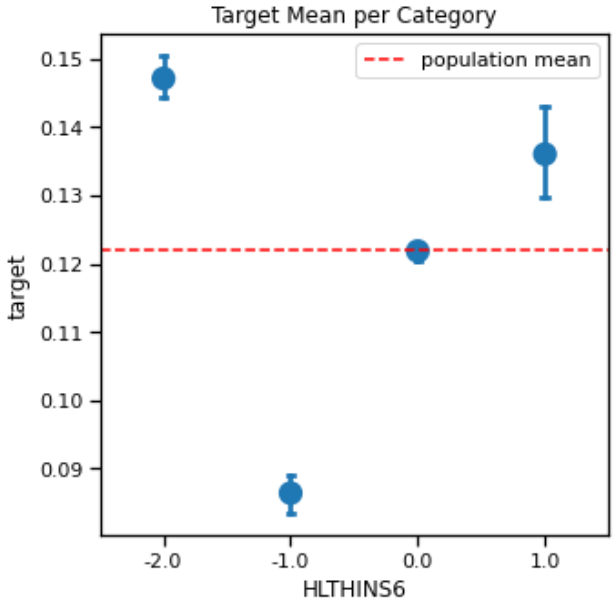

Distribution of HLTHINS4
Count
HLTHINS4

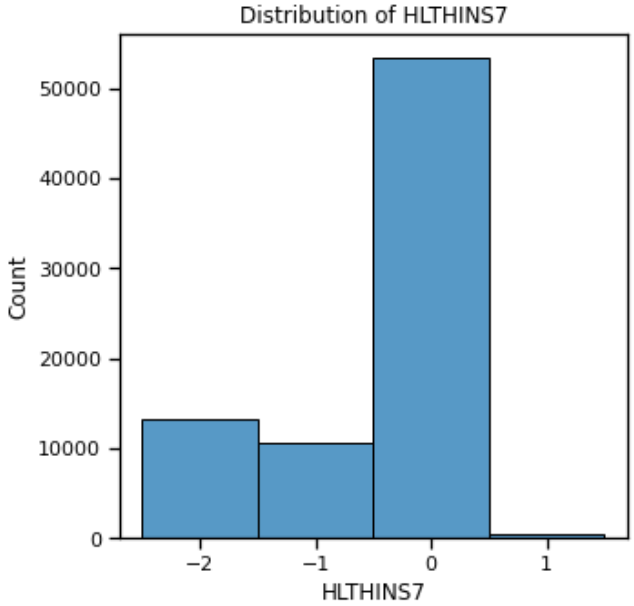

Target Mean per Category
population mean
target
HLTHINS4

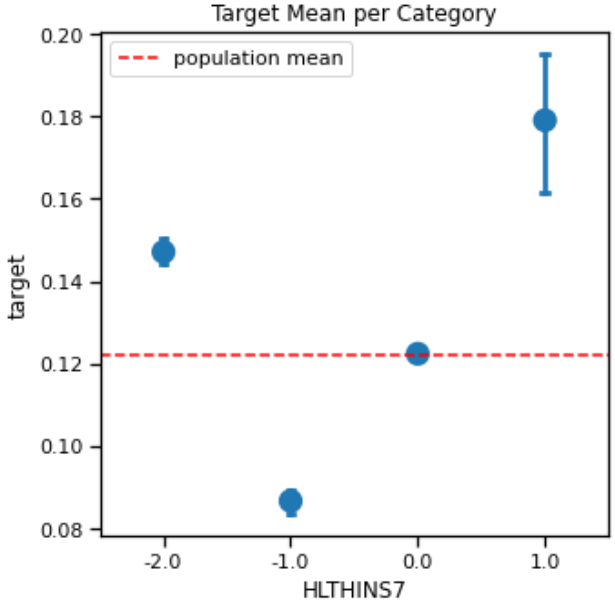

Distribution of HLTHINS5
Count
HLTHINS5
0
10000
20000
30000
40000
50000
−2
−1
0
1

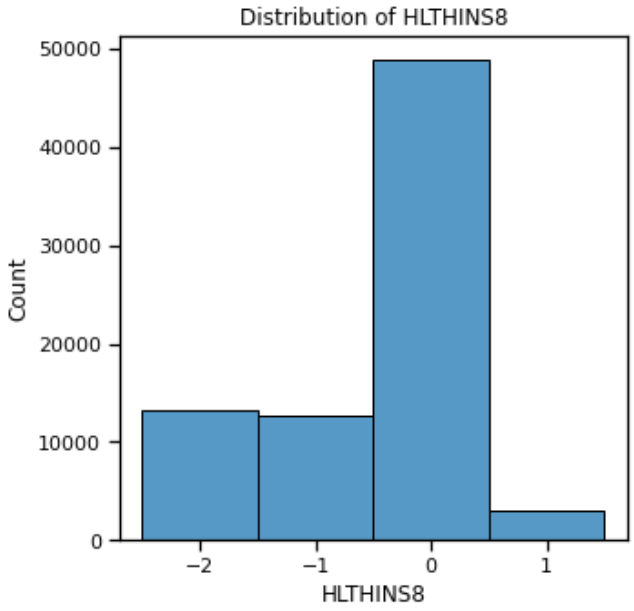

Target Mean per Category
population mean
target
HLTHINS5
0.09
0.10
0.11
0.12
0.13
0.14
0.15
-2.0
-1.0
0.0
1.0

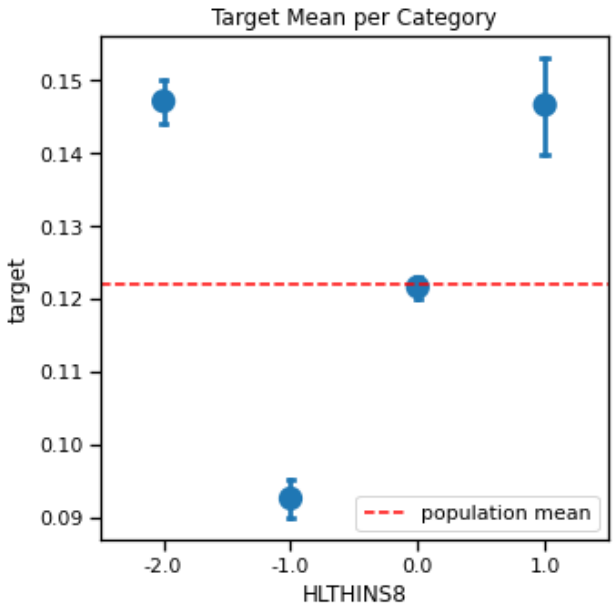

Distribution of HLTHINS6
Count
HLTHINS6
0
10000
20000
30000
40000
50000
−2
−1
0
1

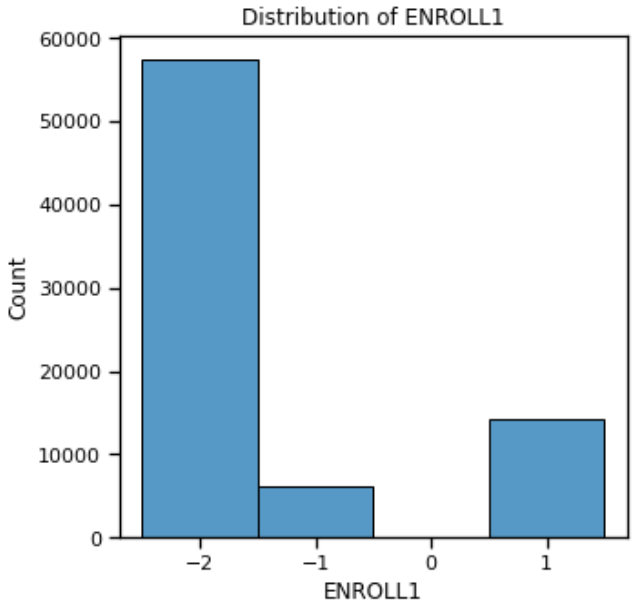

Target Mean per Category
population mean
target
HLTHINS6
0.09
0.10
0.11
0.12
0.13
0.14
0.15
-2.0
-1.0
0.0
1.0

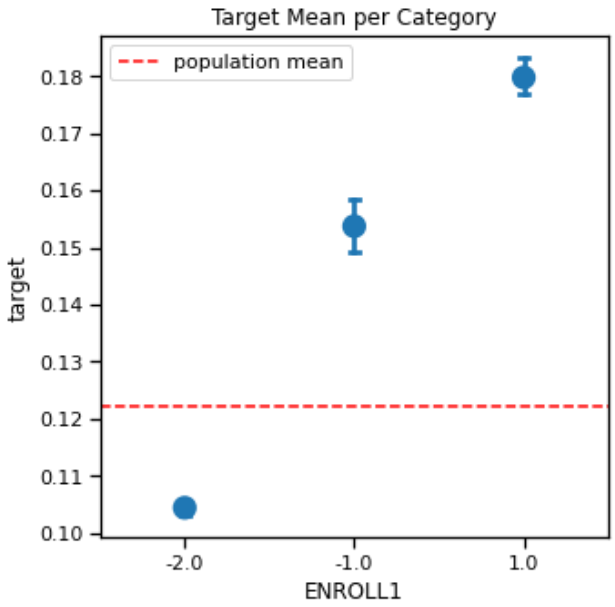

Distribution of HLTHINS7
Count
HLTHINS7
0
10000
20000
30000
40000
50000
−2
−1
0
1

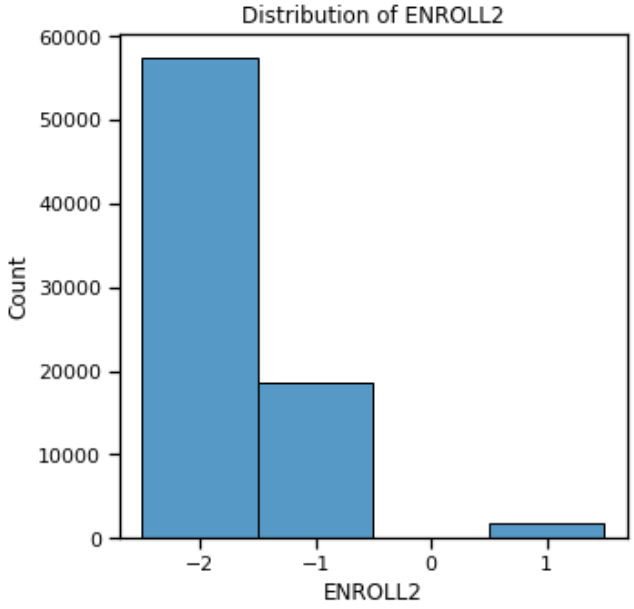

Target Mean per Category
population mean
target
HLTHINS7
0.08
0.10
0.12
0.14
0.16
0.18
0.20
-2.0
-1.0
0.0
1.0

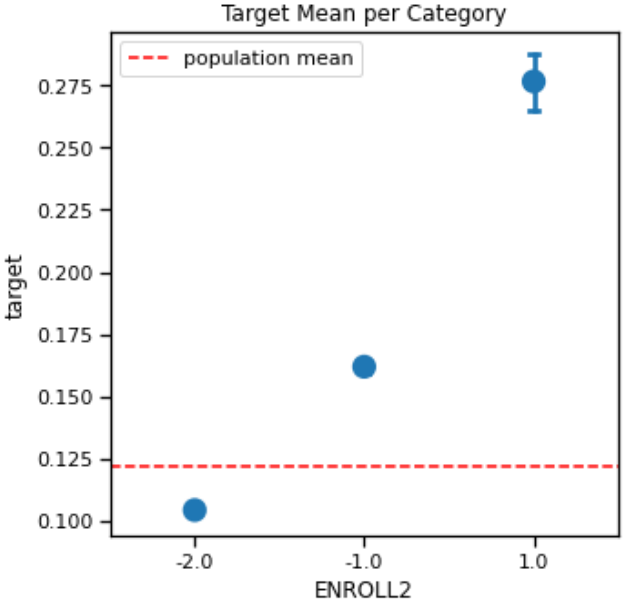

Distribution of HLTHINS8
Count
HLTHINS8

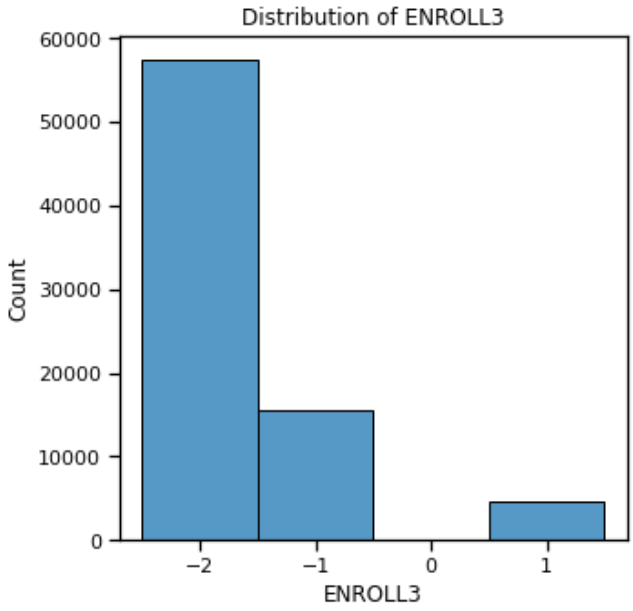

Target Mean per Category
target
HLTHINS8
population mean

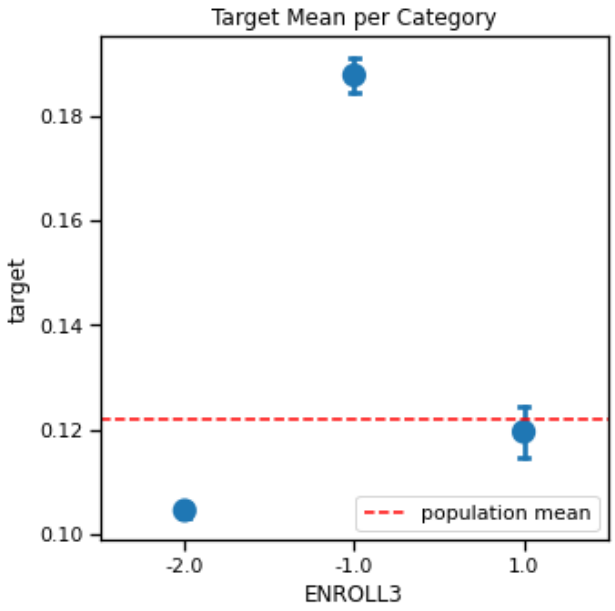

Distribution of ENROLL1
Count
ENROLL1

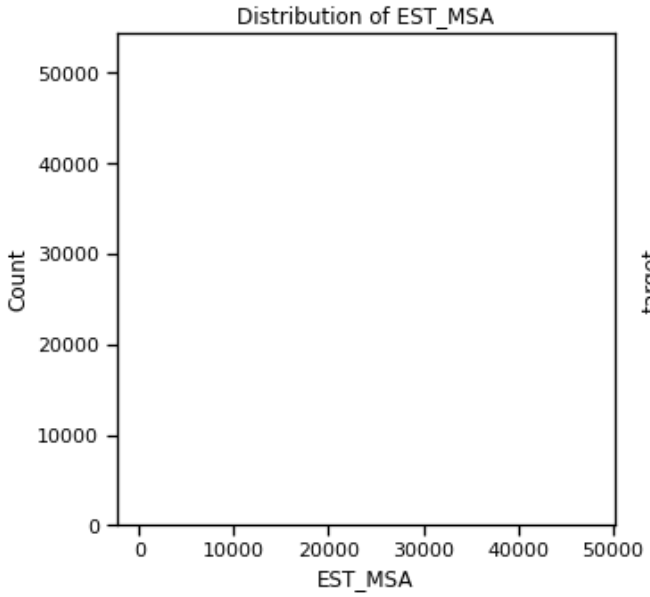

Target Mean per Category
target
ENROLL1
population mean

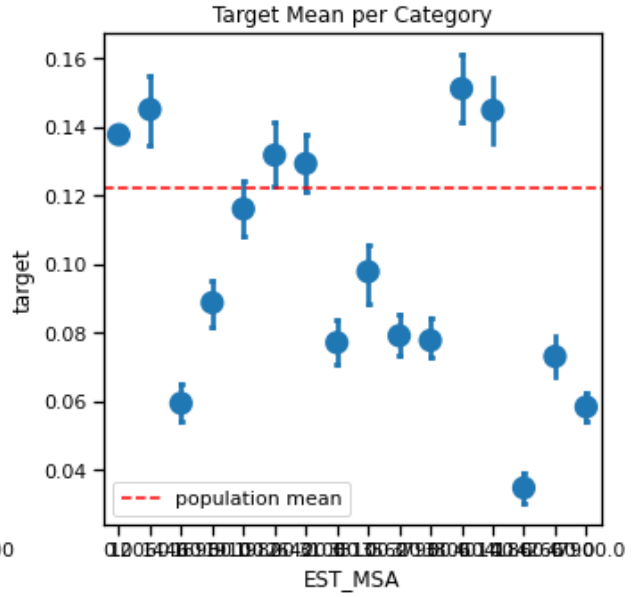

Distribution of ENROLL2
Count
ENROLL2

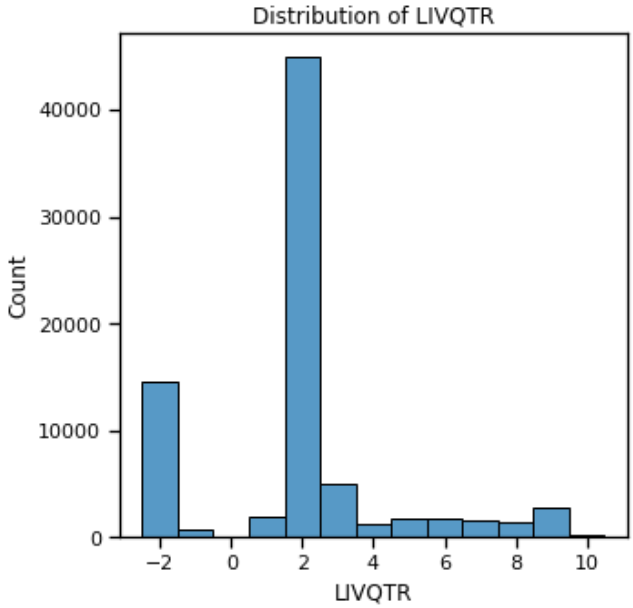

Target Mean per Category
target
ENROLL2
population mean

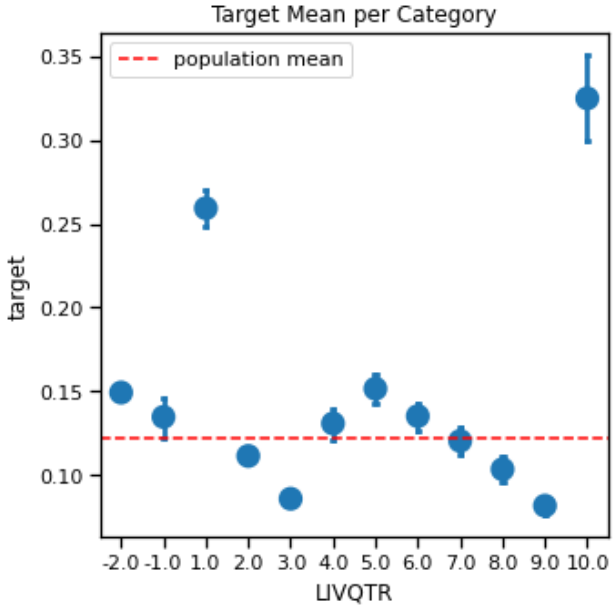

Distribution of ENROLL3
Count
ENROLL3

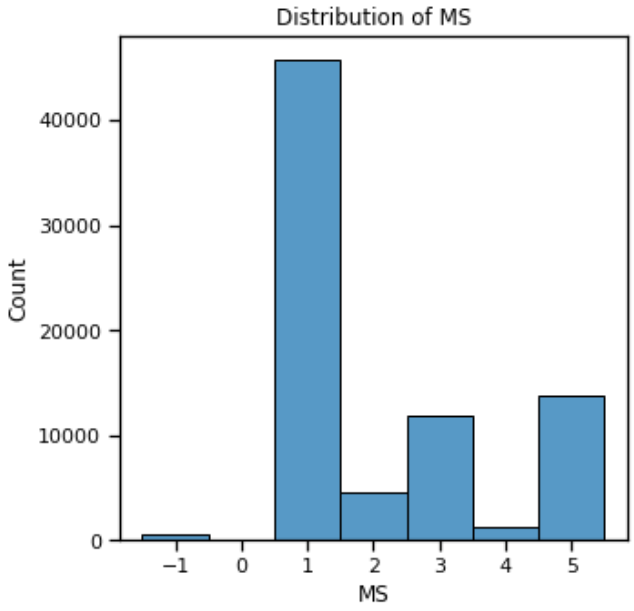

Target Mean per Category
target
ENROLL3
population mean

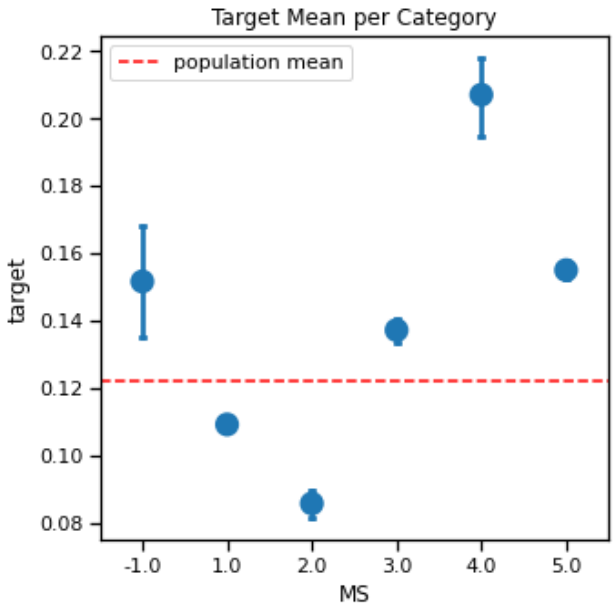

Distribution of EST_MSA
Count
EST_MSA

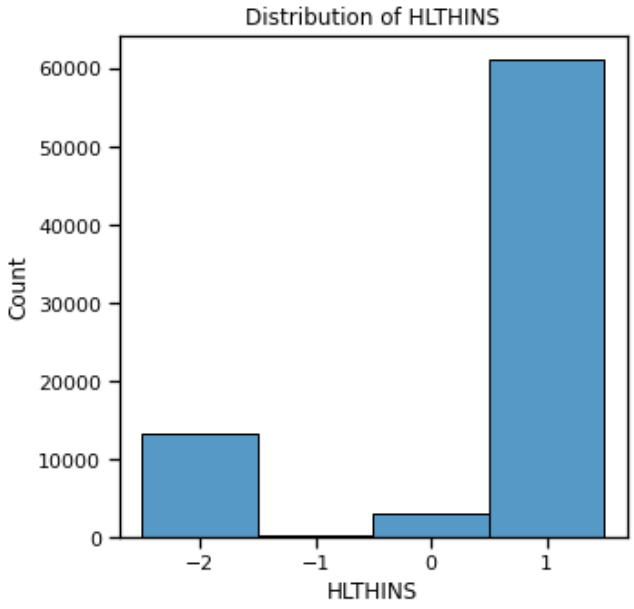

Target Mean per Category
target
EST_MSA
population mean

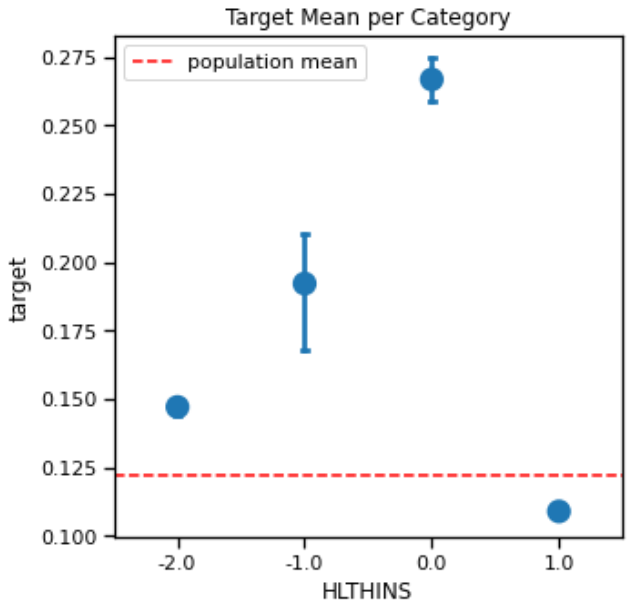

Distribution of LIVQTR
Count
LIVQTR

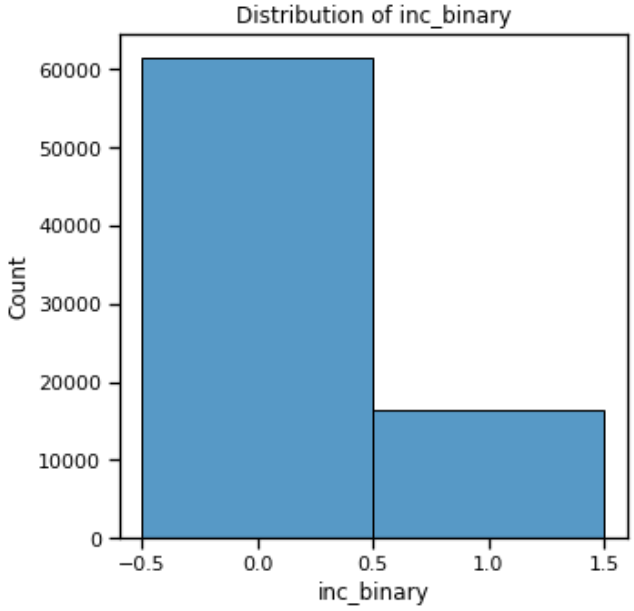

Target Mean per Category
target
LIVQTR
population mean

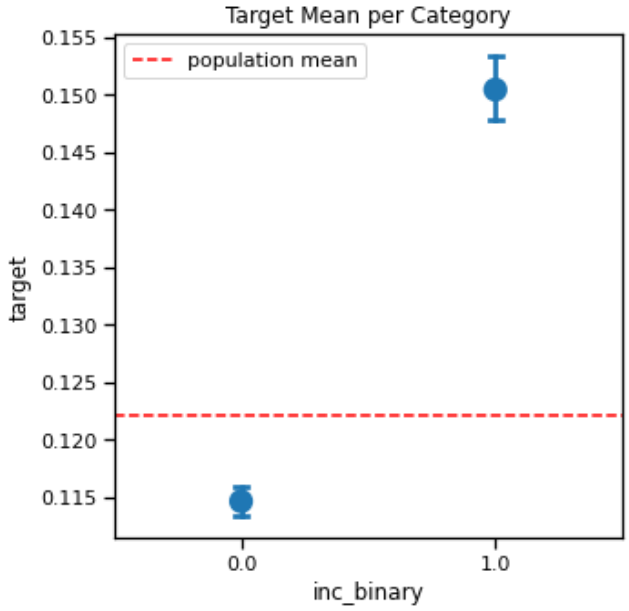

Distribution of MS
Count
MS

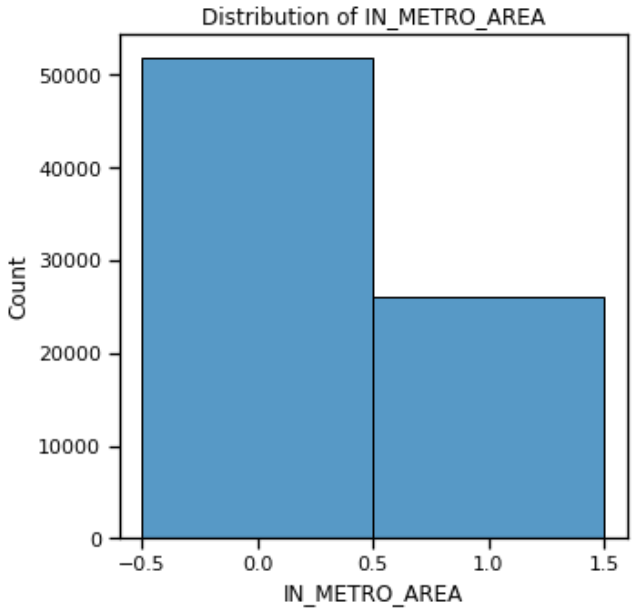

Target Mean per Category
population mean
target
MS

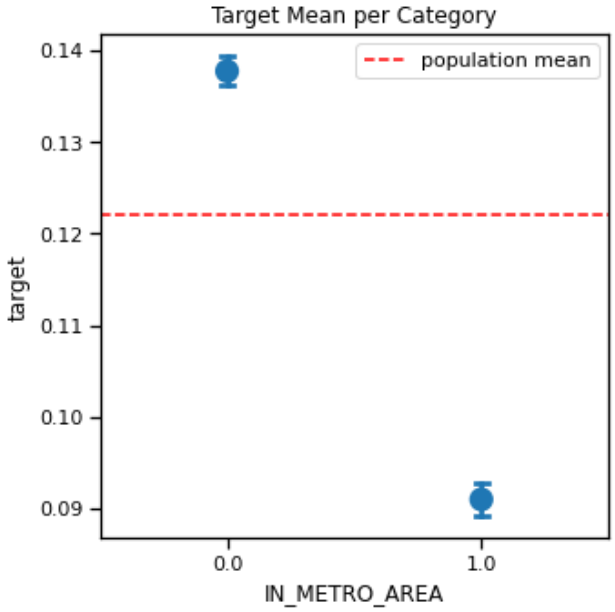

Distribution of HLTHINS
Count
HLTHINS

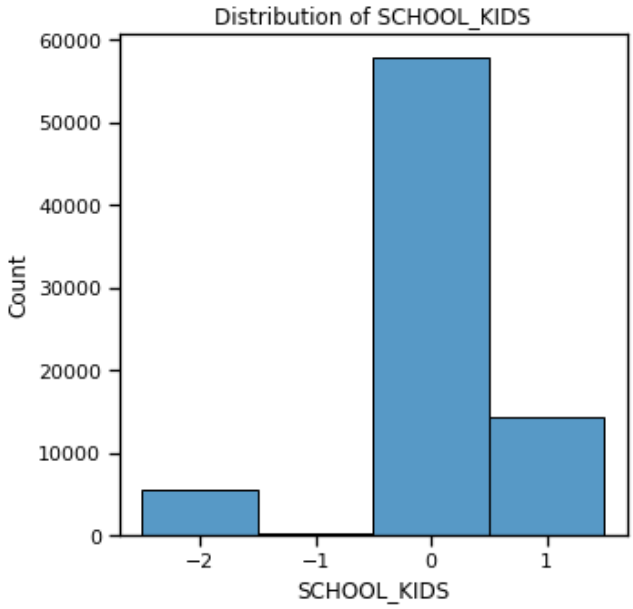

Target Mean per Category
population mean
target
HLTHINS

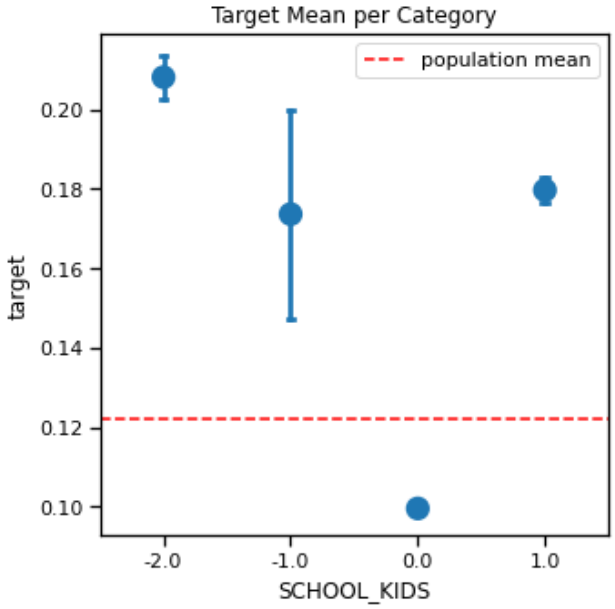

Distribution of inc_binary
Count
inc_binary

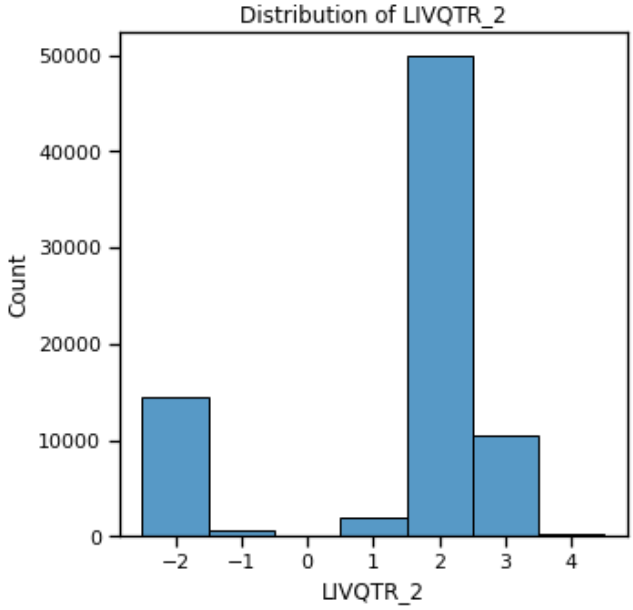

Target Mean per Category
population mean
target
inc_binary

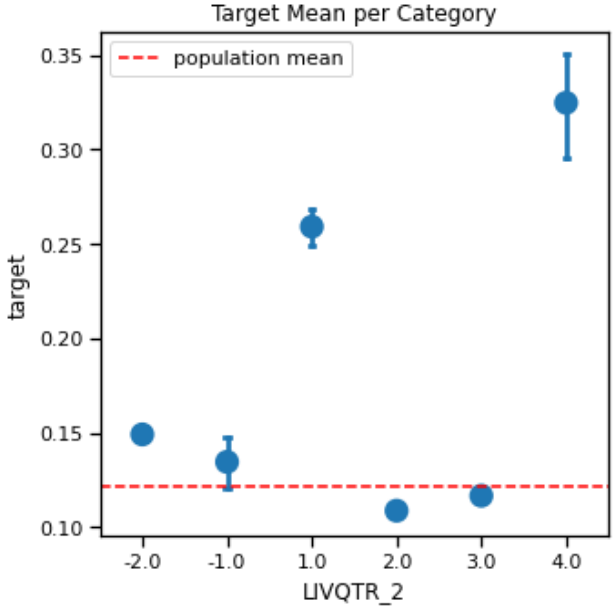

Distribution of IN_METRO_AREA
Count
IN_METRO_AREA

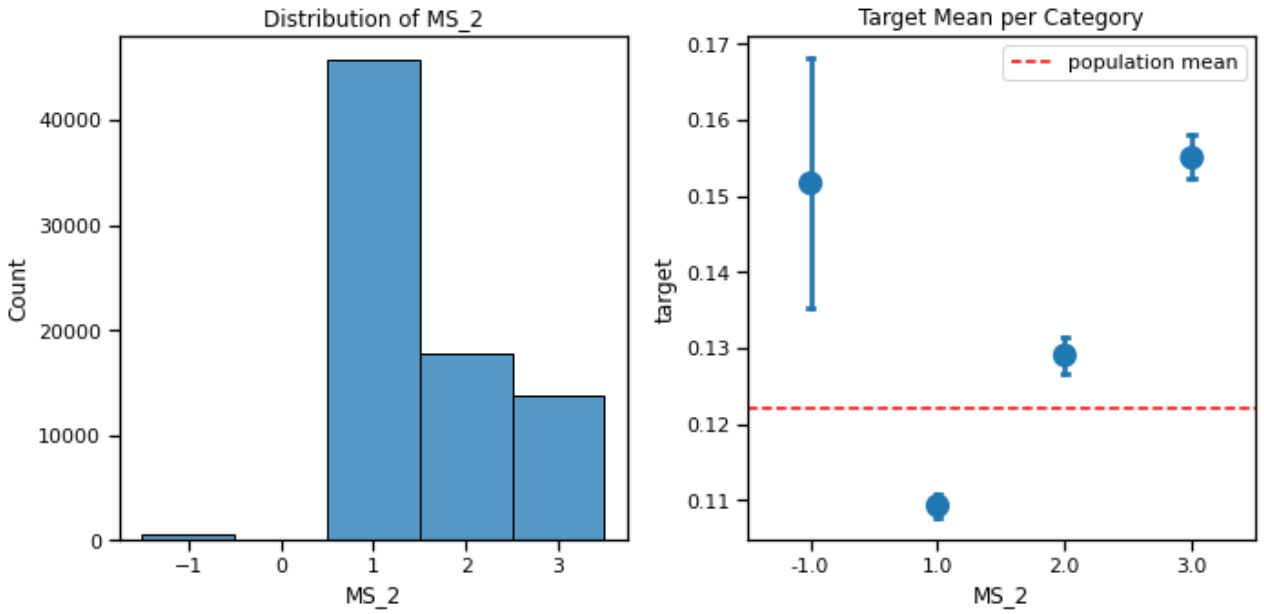

Target Mean per Category
population mean
target
IN_METRO_AREA

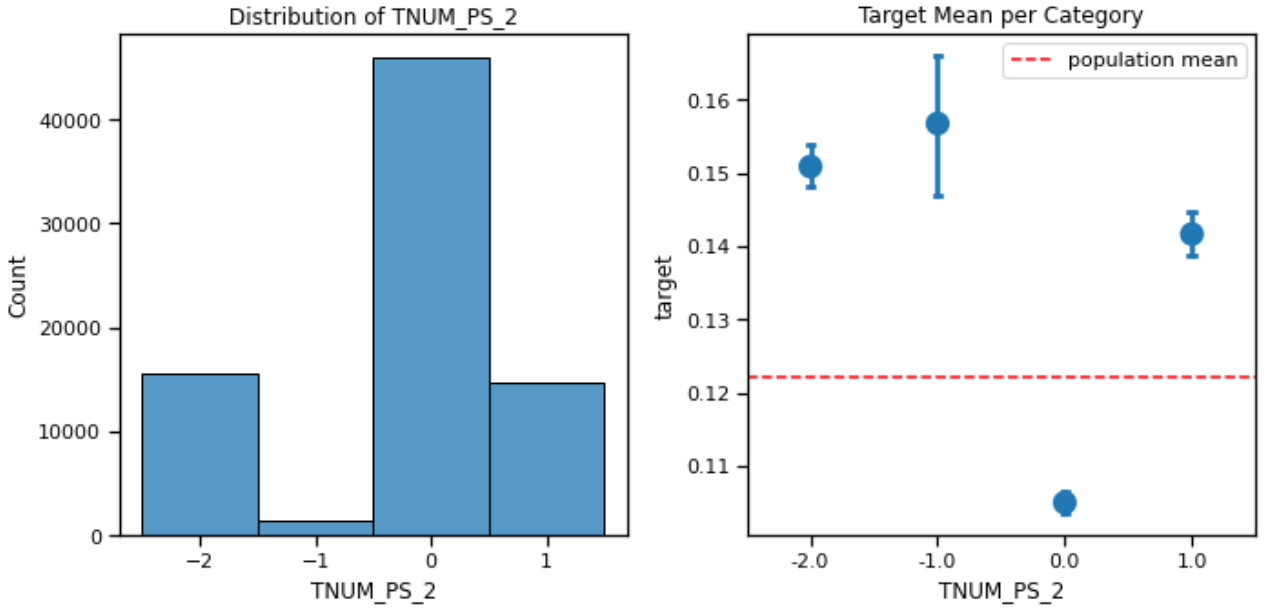

Distribution of SCHOOL_KIDS
Count
SCHOOL_KIDS


Target Mean per Category
population mean
target
SCHOOL_KIDS


Distribution of LIVQTR_2
Count
LIVQTR_2


Target Mean per Category
population mean
target
LIVQTR_2





Chi-Squared for Categorical versus Target

In [349…
```
import scipy.stats as stats
```

In [120…
```
# calculate chi-squared statistic for each categorical predictor with target
# Make a list of any independent variables, which could be removed
# dependent variables are ones we want to keep

ind_cats = []
prob = 0.95

for col in cat_cols + cat_cols_v1 + cat_cols_v2:
    # get the contingency table using crosstab
    crosstab = pd.crosstab(df['target'], df[col])

    # use scipy to get chi2 statistic and other info
    stat, p, dof, expected = stats.chi2_contingency(crosstab)

    #https://machinelearningmastery.com/chi-squared-test-for-machine-learning/
```

```
    # interpret test-statistic
    critical = stats.chi2.ppf(prob, dof)
    # append features that are not dependent on target to list
    if abs(stat) <= critical:
        ind_cats.append([col, stat, p])

ind_cats_df = pd.DataFrame(ind_cats, columns=['feature name', 'chi2 stat', 'p-va
ind_cats_df
```

Out[120…

	feature name	chi2 stat	p-value
0	SSA_APPLY	2.973671	0.395702

Chi-squared test confirmed what I guessed from the visualizations, which is that all the categorical features have a dependent relationship on the target, with the exception of `SSA_APPLY`.

In [351…

```
# RRACE category labels are:
#1) White, Alone
#2) Black, Alone
#3) Asian, Alone
#4) Any other race alone, or race in combination

# Are the proportions of respondents similar to population proportions?
df['RRACE'].value_counts(normalize=True)
```

Out[351…

```
1.0    0.825609
2.0    0.078073
3.0    0.050434
4.0    0.045885
Name: RRACE, dtype: float64
```

In [352…

```
df['RHISPANIC'].value_counts(normalize=True)
```

Out[352…

```
0.0    0.904953
1.0    0.095047
Name: RHISPANIC, dtype: float64
```

The 2020 Census statistics report that the US population is about 76% White, 13% Black, and about 6% Asian. About 18% of the population identifies as Hispanic or Latinx.

In this data, White respondents are overrepresented, Black respondents are far underrepresented, Asian respondents are slightly underrepresented, and Hispanic/Latinx respondents are far underrepresented.

Focusing on the actual answers (values of 0 and above):

- `RRACE` has few respondents who are not white. In this data, White respondents are overrepresented, Black and Hispanic/Latinx respondents are far underrepresented, and Asian respondents are slightly underrepresented.
- `SPNDSRC8` value of 1 has a much higher target mean than the population. Looking at that question, it's related to whether the respondent used SNAP funds in the past 7 days. Since I already have a column to indicate SNAP participation, I'll drop this column.

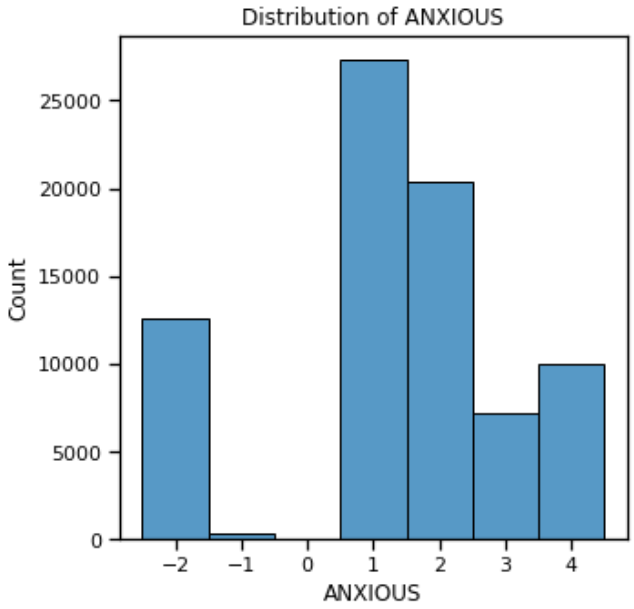
Focusing on the -2 and -1 answers, few respondents opted not to answer any given question. I'm considering whether to do something similar to what I did with Incomplete where I create a

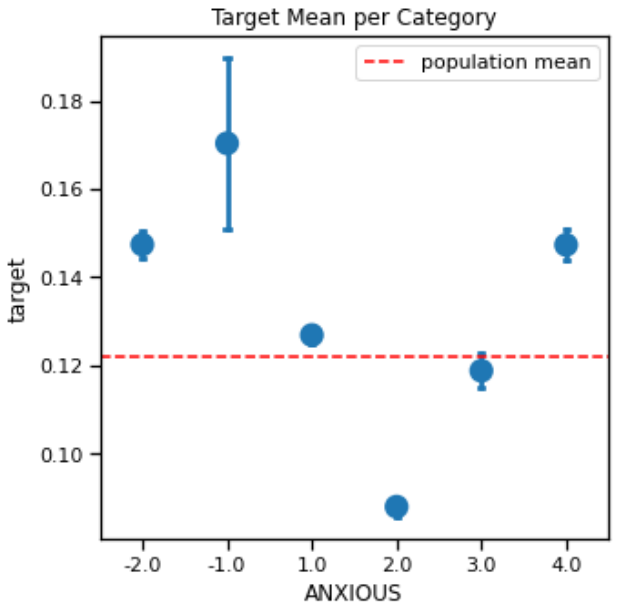
single column to represent whether the respondent skipped ANY questions.

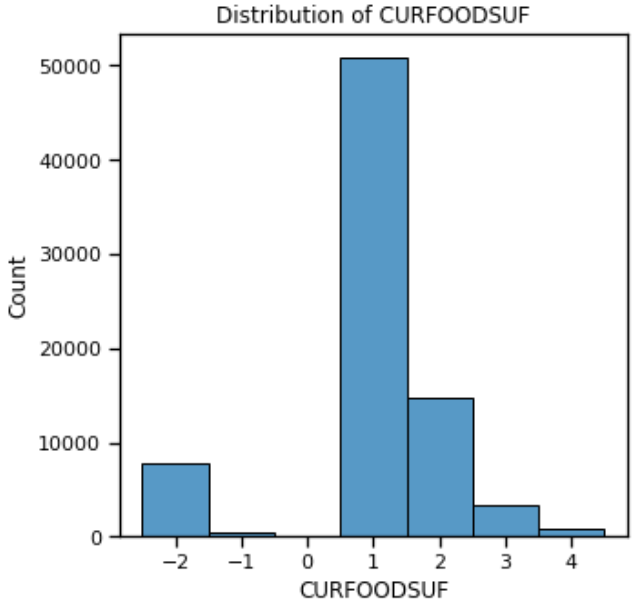
In [353…
```
# drop `SPNDSRC8`
cat_cols.remove('SPNDSRC8')
```

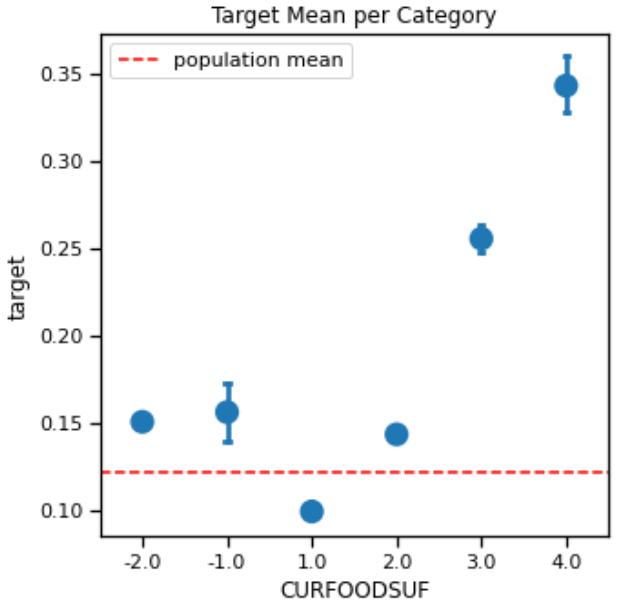
Ordinal Categorical Features

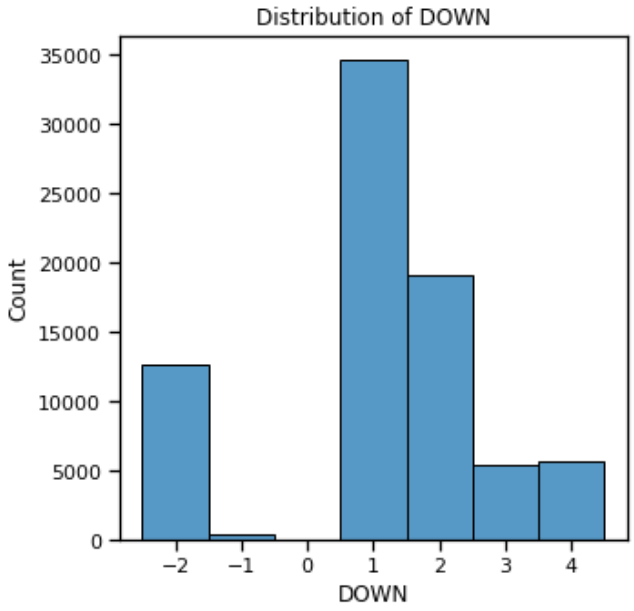
In [354…
```
# explore ordinal categorical columns
if rerun_viz:
    dstools.explore_data_catbin(ord_cols + ['THHLD_NUMADLT', 'THHLD_NUMKID'],
                                df, 'target')
```

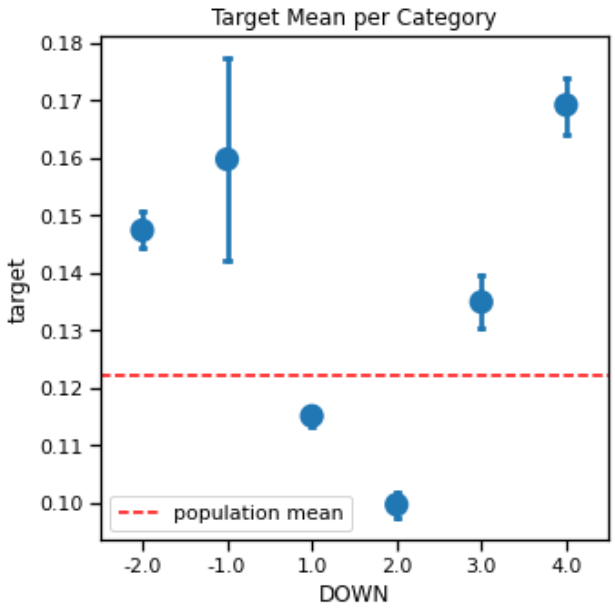


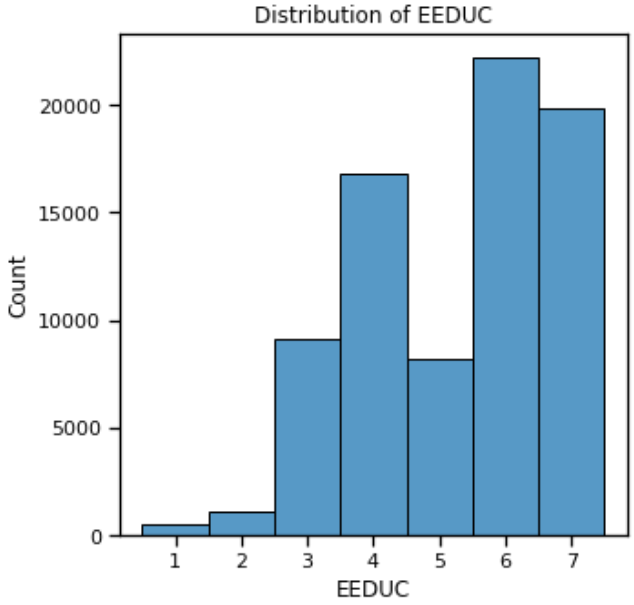


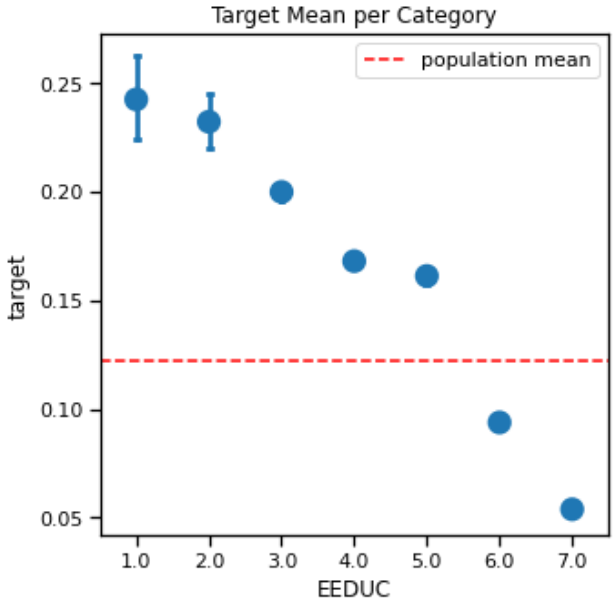


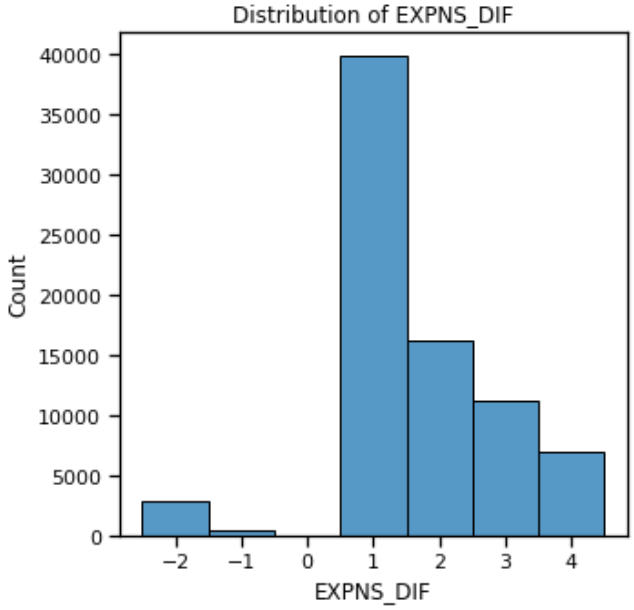


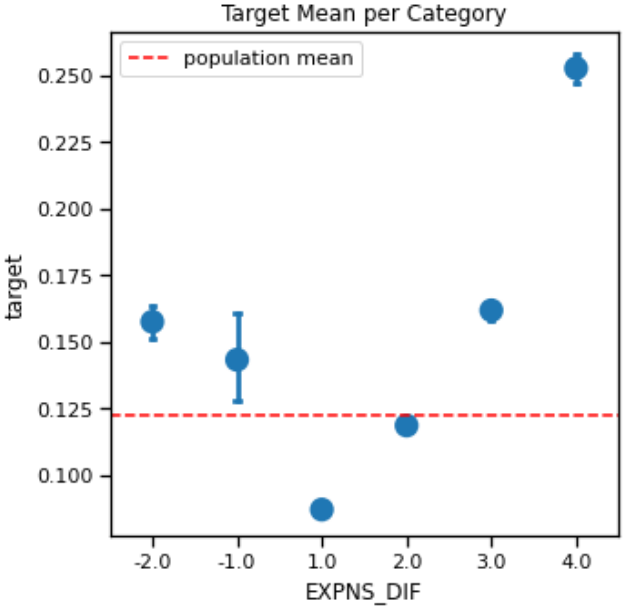

Distribution of DOWN
Count
DOWN

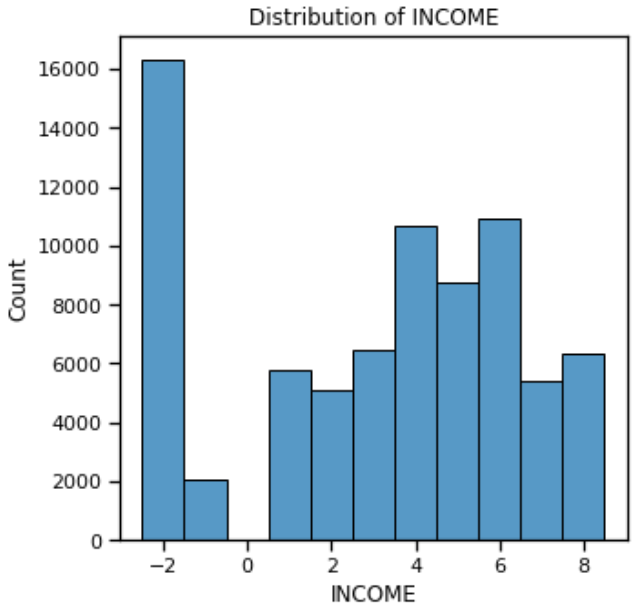

Target Mean per Category
target
DOWN
population mean

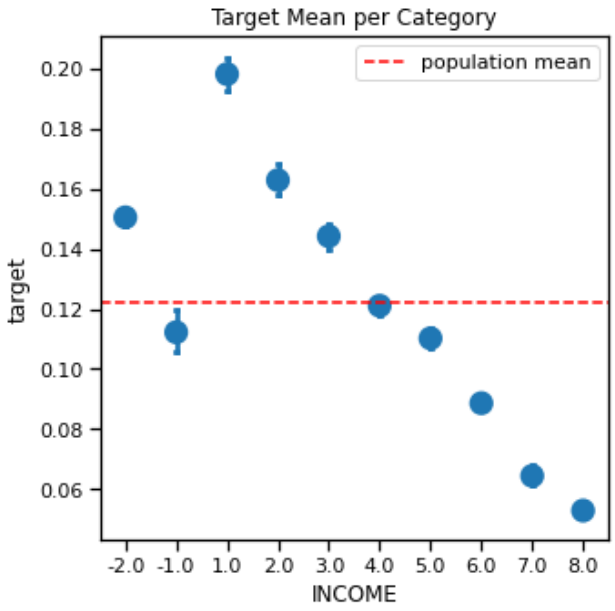

Distribution of EEDUC
Count
EEDUC

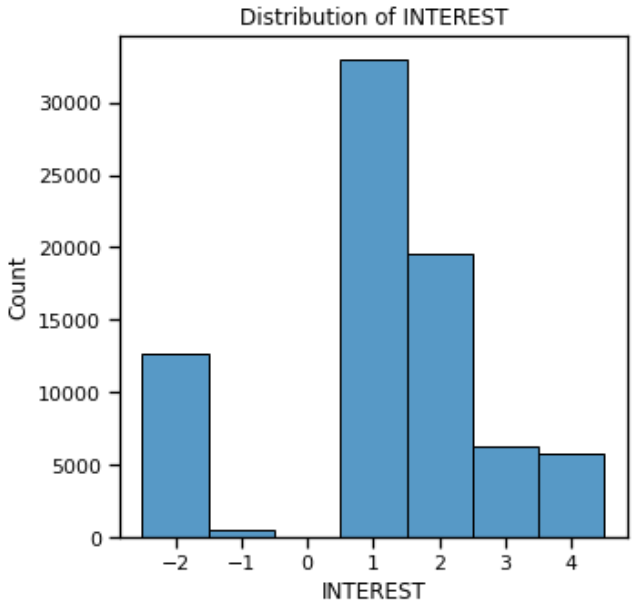

Target Mean per Category
target
EEDUC
population mean

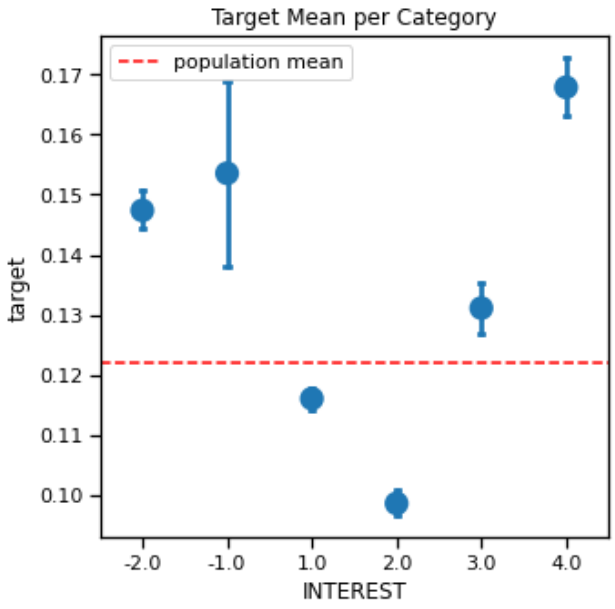

Distribution of EXPNS_DIF
Count
EXPNS_DIF

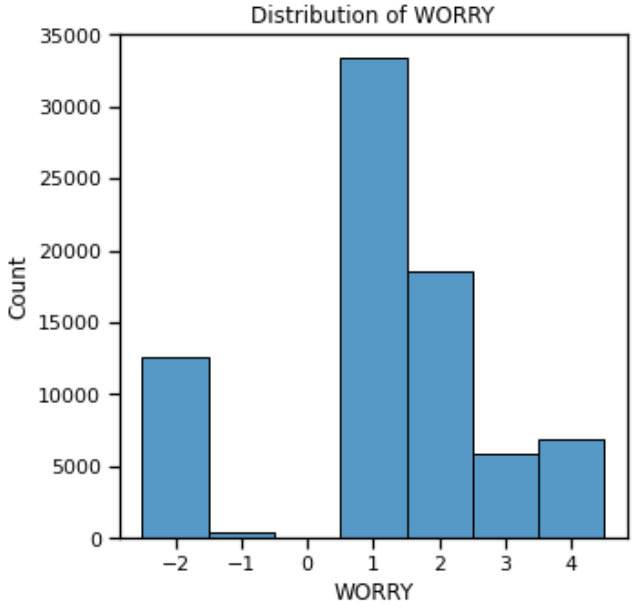

Target Mean per Category
target
EXPNS_DIF
population mean

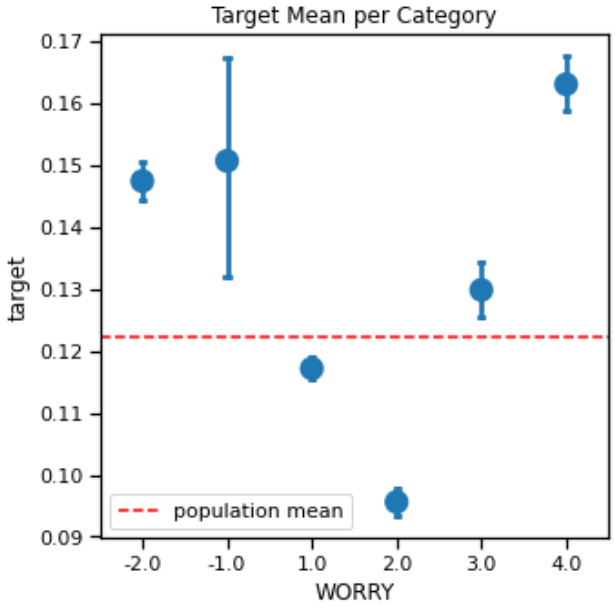

Distribution of INCOME
Count
INCOME

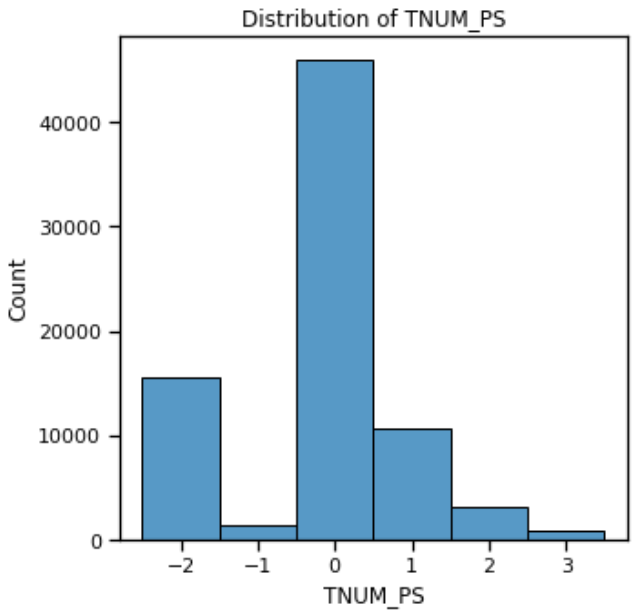

Target Mean per Category
population mean
target
INCOME

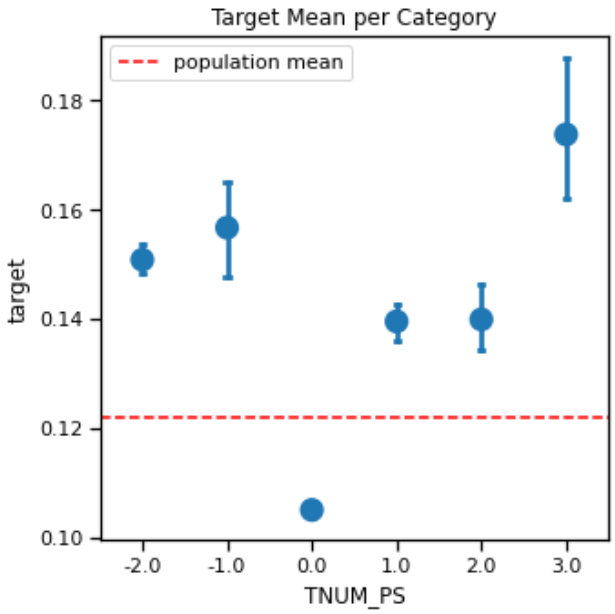

Distribution of INTEREST
Count
INTEREST

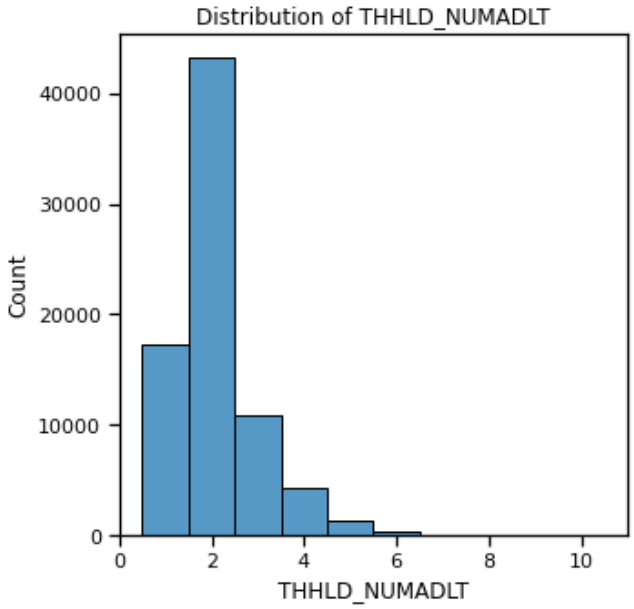

Target Mean per Category
population mean
target
INTEREST

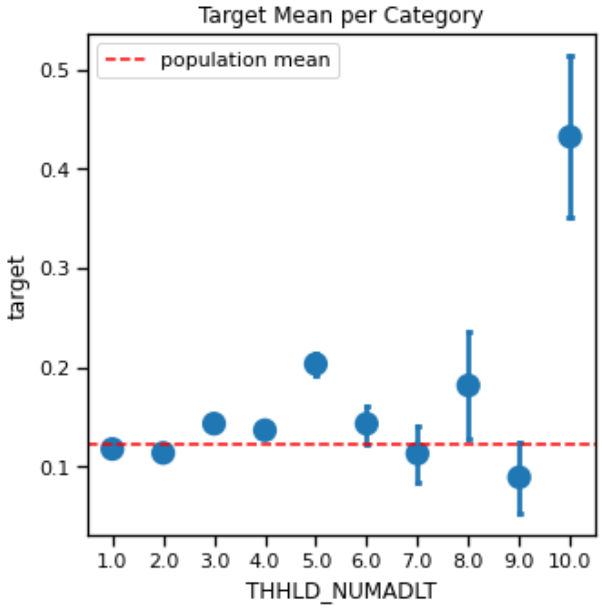

Distribution of WORRY
Count
WORRY

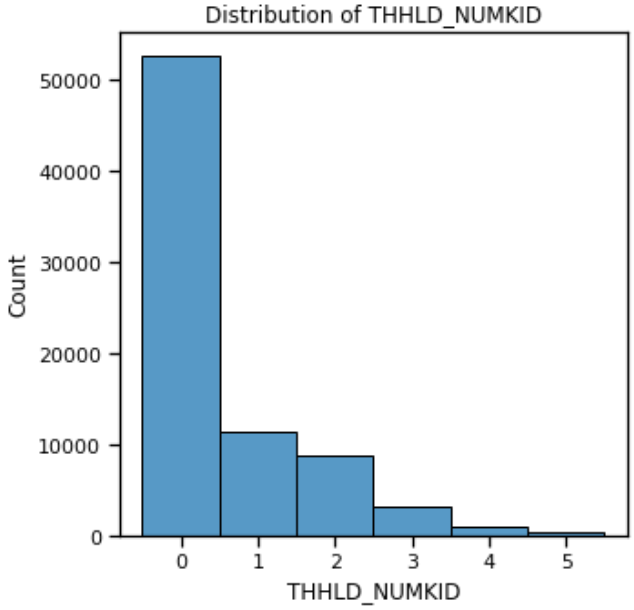

Target Mean per Category
population mean
target
WORRY

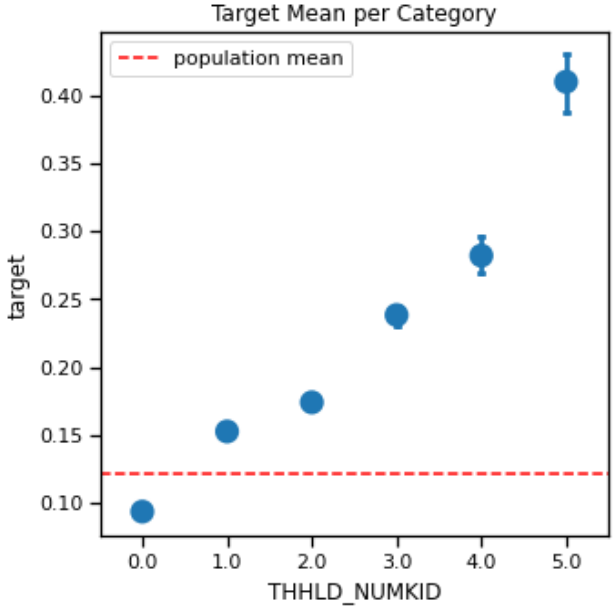












Focusing on the actual answers (values of 0 and above):

- `ANXIOUS`, `DOWN`, `INTEREST` and `WORRY` all share a similar pattern.
  - Responses of 1 (Not at All) are fairly close to the population mean but a bit below
  - Responses of 2 (Several Days) are well below population mean
  - Responses of 3 (More than half the days) are again fairly close to the population mean but a bit above
  - Responses of 4 (Nearly every day) are well above the population mean

  I'm not sure I can treat these as numbers, I think I have to OHE them.

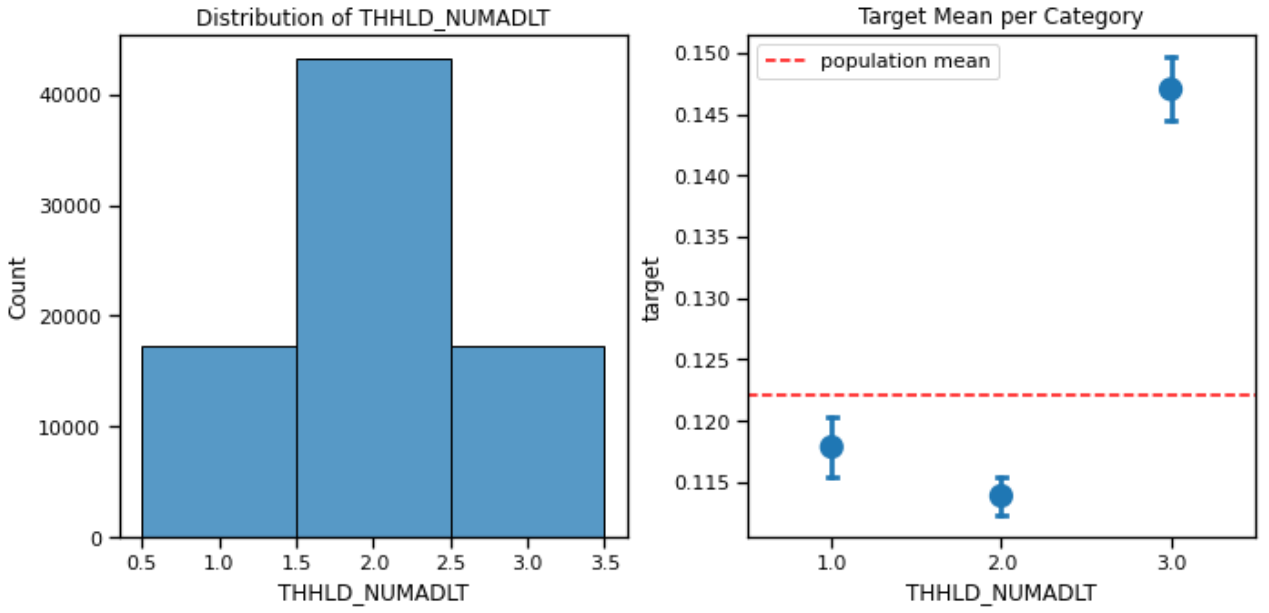
- `THHLD_NUMADLT` doesn't appear to have much of a linear relationship with target mean. Most values are pretty close to the mean, and those that appear to deviate have very few observations. I'm going to keep this column, but consolidate the number of adults above 3 with 3.

- The rest of the features do show a linear relationship with the target mean, so I will treat them as numeric columns.

In [355…
```
# Combine THHLD_NUMADLT > 3 with 3
df.loc[df['THHLD_NUMADLT'] > 3, 'THHLD_NUMADLT'] = 3
df['THHLD_NUMADLT'].value_counts()
```

Out[355…
```
2.0    43223
1.0    17308
3.0    17294
Name: THHLD_NUMADLT, dtype: int64
```

In [356…
```
# review THHLD_NUMADLT's new distribution
dstools.explore_data_catbin(['THHLD_NUMADLT'], df, 'target')
```



In [385…
```
# Reorganize my lists of columns by type, based on what I've learned and updated
cat_cols = cat_cols + ['ANXIOUS', 'DOWN', 'INTEREST', 'WORRY', 'THHLD_NUMADLT']

for col in ['ANXIOUS', 'DOWN', 'INTEREST', 'WORRY']:
    ord_cols.remove(col)
```

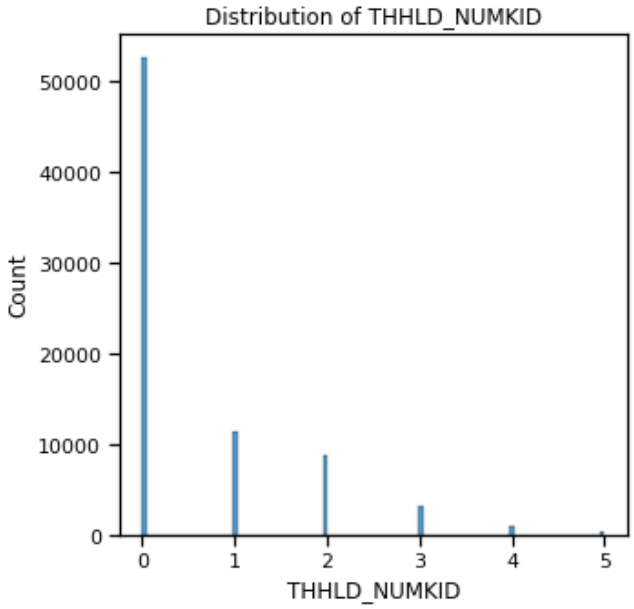
```
num_cols.remove('THHLD_NUMADLT')

print(cat_cols)
print(ord_cols)
print(num_cols)
```

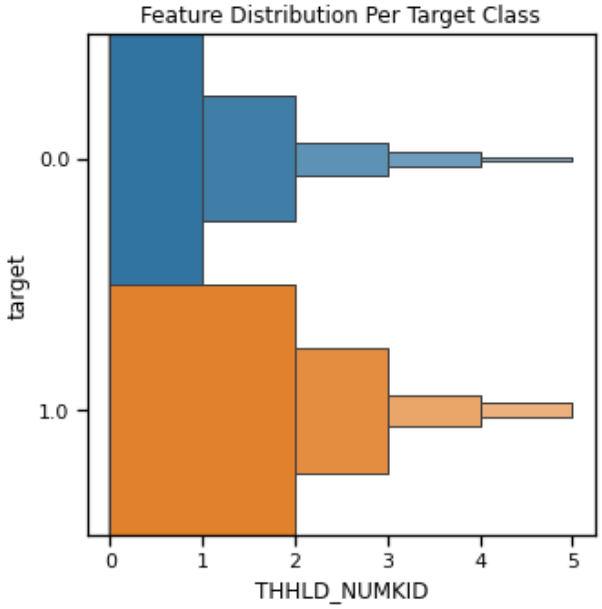
```
['ANYWORK', 'CHNGHOW1', 'CHNGHOW10', 'CHNGHOW11', 'CHNGHOW12', 'CHNGHOW2', 'CHNG
HOW3', 'CHNGHOW4', 'CHNGHOW5', 'CHNGHOW6', 'CHNGHOW7', 'CHNGHOW8', 'CHNGHOW9',
'DELAY', 'EGENDER', 'EIP', 'EXPCTLOSS', 'FEWRTRANS', 'FEWRTRIPS', 'FREEFOOD', 'H
ADCOVID', 'MH_NOTGET', 'MH_SVCS', 'NOTGET', 'PLNDTRIPS', 'PRESCRIPT', 'RHISPANI
C', 'RRACE', 'SNAP_YN', 'SPNDSRC1', 'SPNDSRC2', 'SPNDSRC3', 'SPNDSRC4', 'SPNDSRC
5', 'SPNDSRC6', 'SPNDSRC7', 'SPNDSRC8', 'SSA_APPLY', 'SSA_RECV', 'TENURE', 'TW_S
TART', 'UI_APPLY', 'WRKLOSS', 'ANXIOUS', 'DOWN', 'INTEREST', 'WORRY', 'THHLD_NUM
ADLT']
['CURFOODSUF', 'EEDUC', 'EXPNS_DIF', 'INCOME']
['THHLD_NUMKID', 'PROP_FOODSPEND_HOME', 'AGE']
```

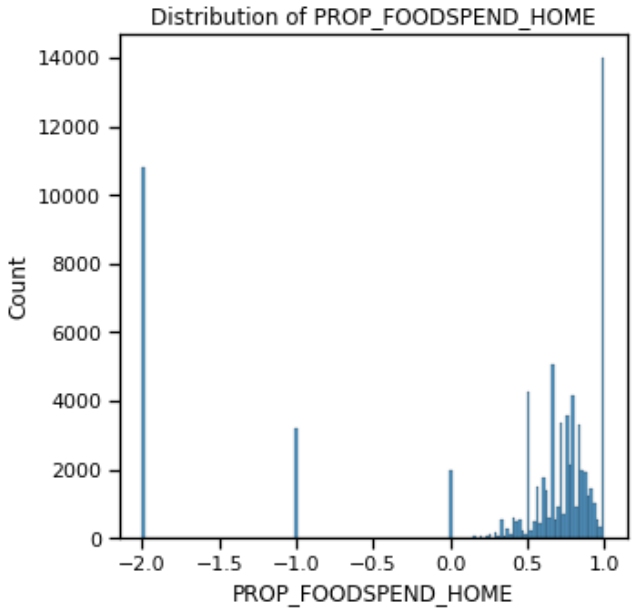
Numeric Features

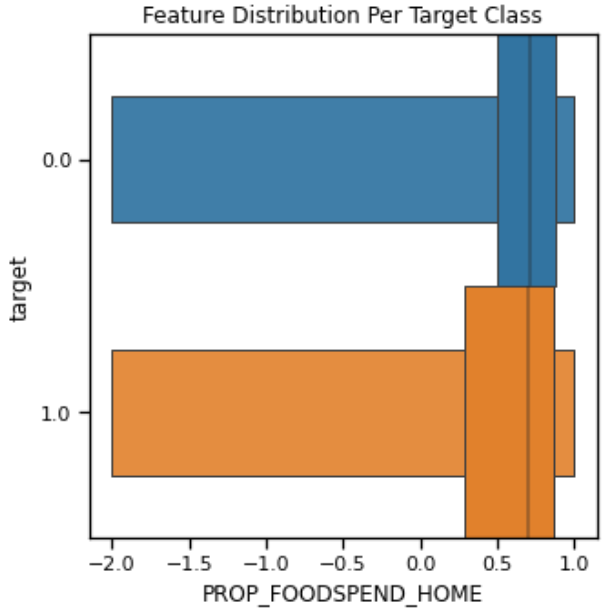
In [386…

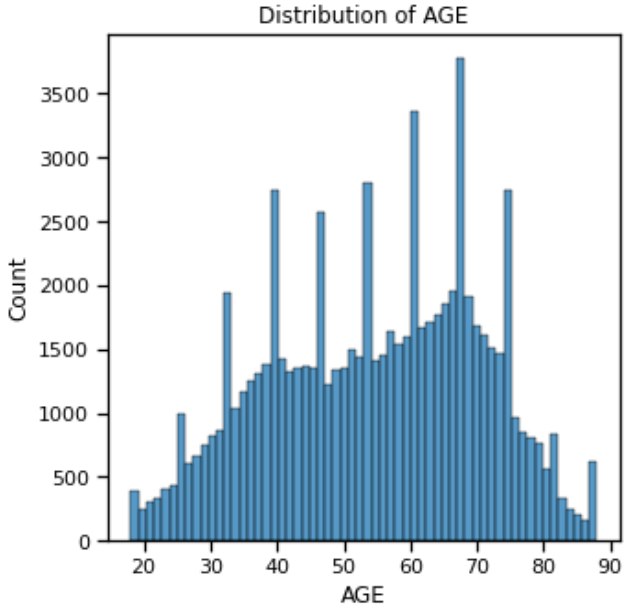
```
if rerun_viz:
    dstools.explore_data_catbin(num_cols + num_cols_v1 + num_cols_v2, df,
                                'target', pred_type='cont')
```

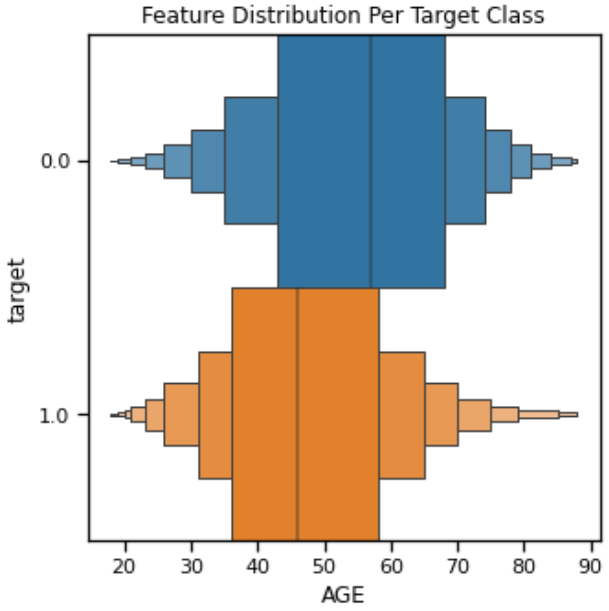


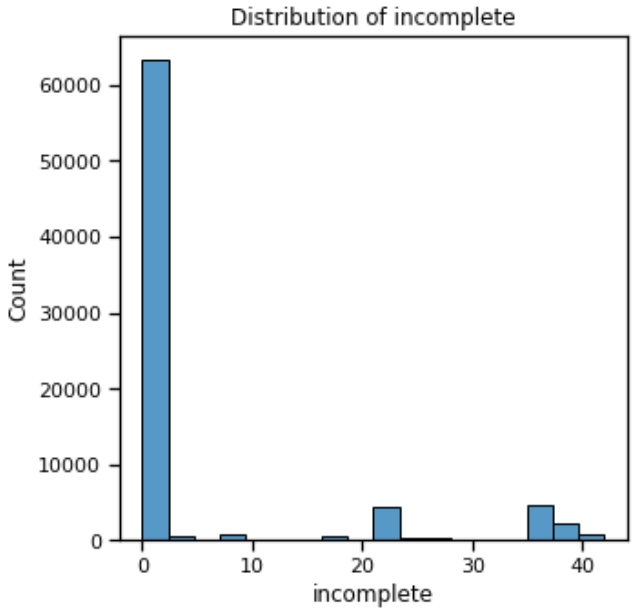


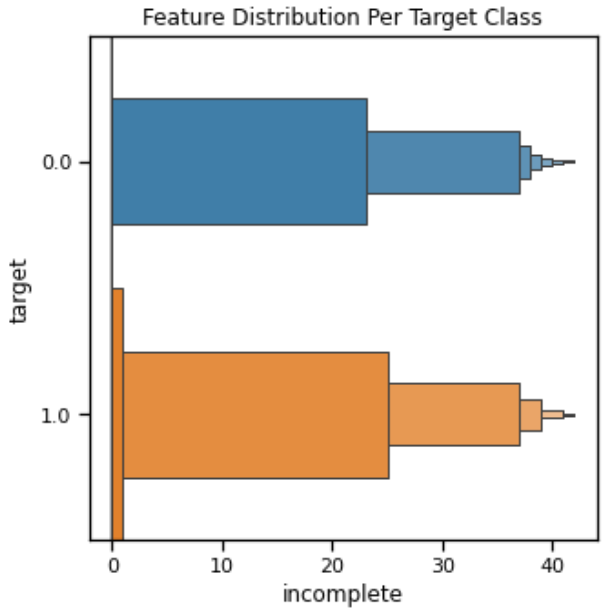

Distribution of PROP_FOODSPEND_HOME
Count
PROP_FOODSPEND_HOME

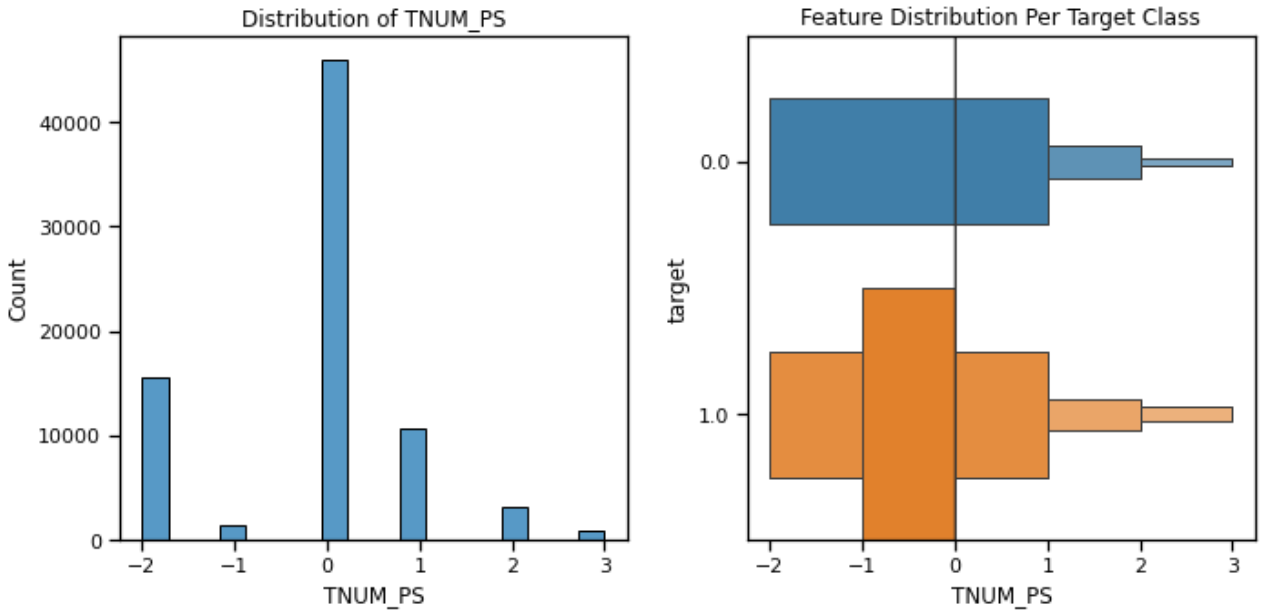

Feature Distribution Per Target Class
target
PROP_FOODSPEND_HOME

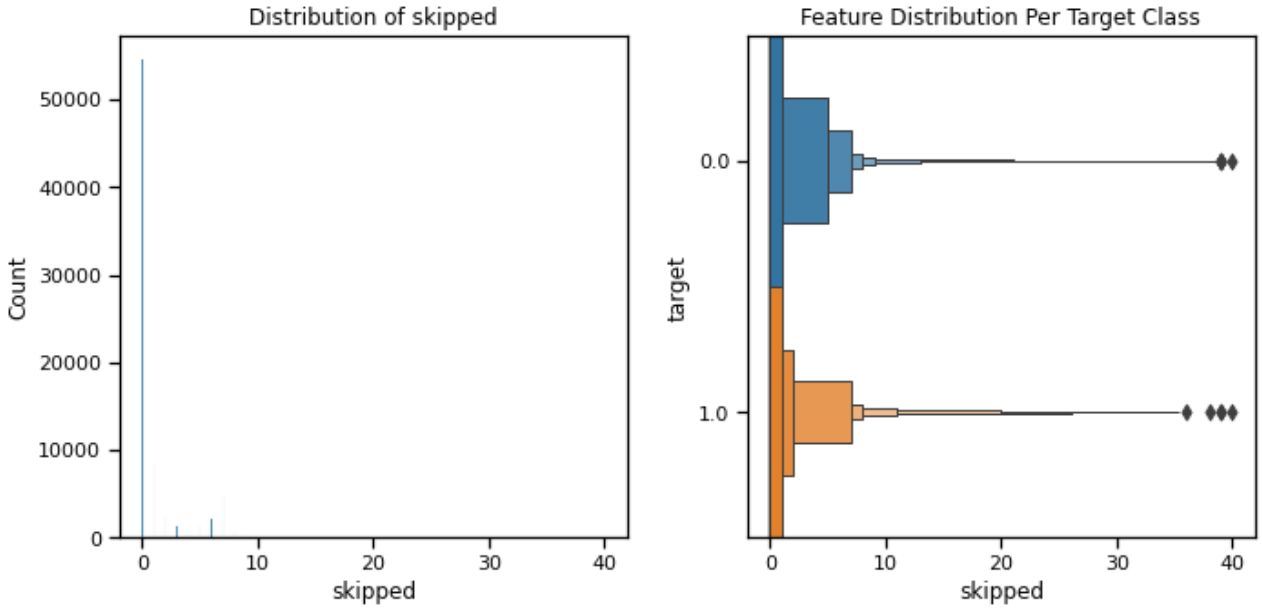

Distribution of AGE
Count
AGE


Feature Distribution Per Target Class
target
AGE


Distribution of incomplete
Count
incomplete


Feature Distribution Per Target Class
target
incomplete





Looking at AGE, I see the histogram is not smooth, but rather appears to have spikes in certain areas. After reviewing the data dictionary again, I notice there are some columns indicating imputed values for some features. I'd like to understand the proportion of imputed values.

In [359…

```
imputed_ind = ['ABIRTH_YEAR','AGENDER','AHISPANIC','ARACE','AEDUC',
               'AHHLD_NUMPER','AHHLD_NUMKID']

for col in imputed_ind:
    print(np.round(df[col].value_counts(normalize=True) * 100, 4))
    print()
```

```
2.0    99.5027
1.0     0.4973
Name: ABIRTH_YEAR, dtype: float64

2.0    99.6646
1.0     0.3354
Name: AGENDER, dtype: float64

2.0    97.8901
1.0     2.1099
```

```
Name: AHISPANIC, dtype: float64

2.0    97.903
1.0     2.097
Name: ARACE, dtype: float64

2.0    99.7019
1.0     0.2981
Name: AEDUC, dtype: float64

2.0    96.6553
1.0     3.3447
Name: AHHLD_NUMPER, dtype: float64

2.0    98.4041
1.0     1.5959
Name: AHHLD_NUMKID, dtype: float64
```

None of the percentages of imputed values look too large. I'm comfortable with this.

Focusing on the actual answers (values of 0 and above):

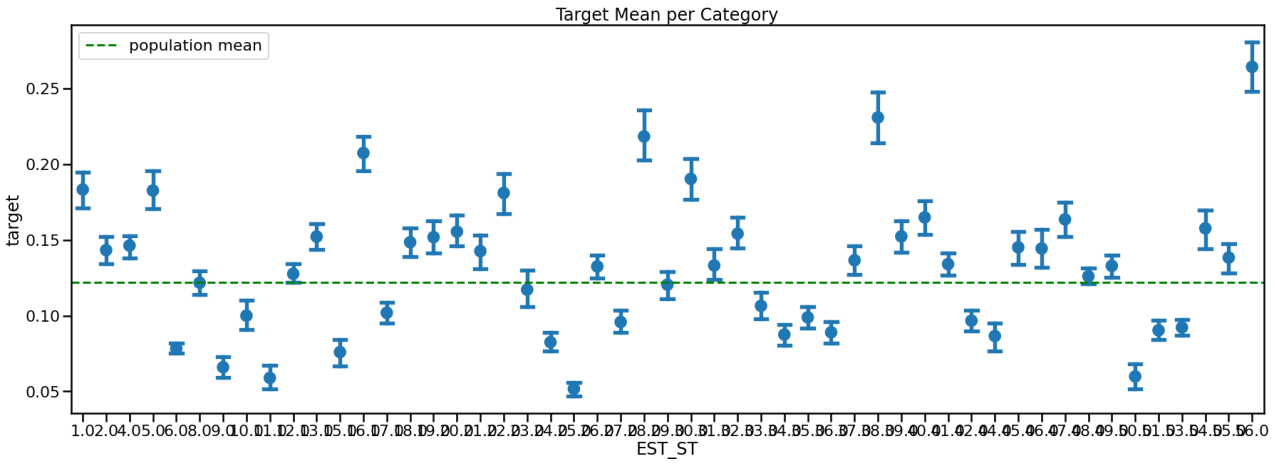
- I don't see much difference in feature distribution of `PROP_FOODSPEND_HOME` for the different target classes

Exploring States

I initially chose to leave out the State, as it has high cardinality and I'd prefer to use the other features such as in metro area instead. But perhaps a visualization will help me decide if I want to try to include it somehow.

In [360…

```
# plot mean of target for each state and compare against target mean for whole s
pop_mean = np.round(df['target'].mean(), 4)

with sns.plotting_context(context='poster'):

    fig, ax = plt.subplots(figsize=(30, 10))

    sns.pointplot(data=df, x='EST_ST', y='target',
                  ci=68, ax=ax, join=False, scale=1, capsize=0.5)
    ax.set_title("Target Mean per Category")
    ax.axhline(pop_mean, color='green', ls='dashed', label='population mean')
    ax.legend();
```



In [361…

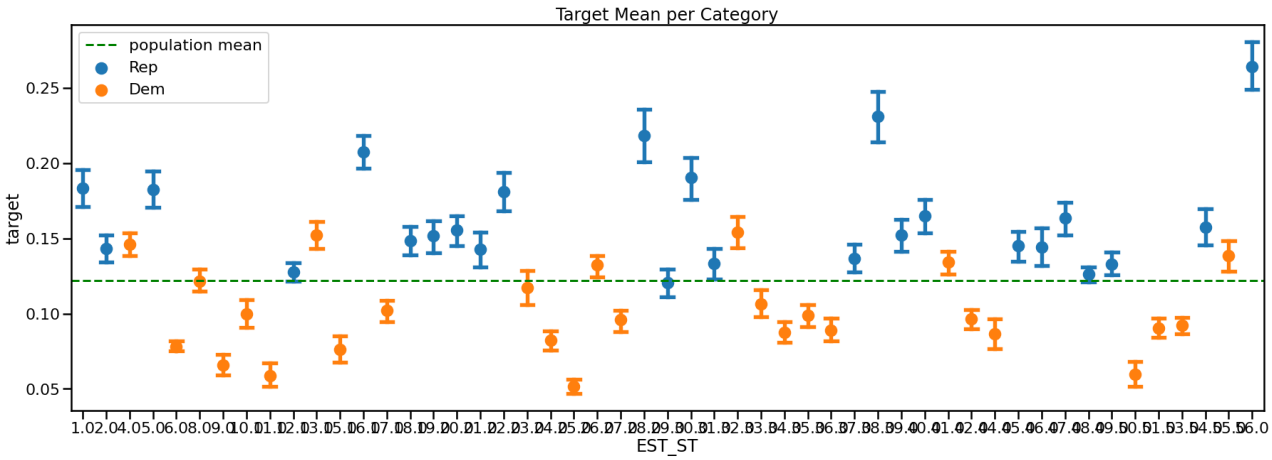
```
# using data from https://www.archives.gov/electoral-college/2020,
# separate states based on the political party to which they allocated
# the majority of their electoral college votes in the 2020 presidential
# election

# data dictionary maps state codes to names
dems = [4, 6, 8, 9, 10, 11, 13, 15, 17, 23, 24, 25, 26, 27, 32, 33, 34, 35,
        36, 41, 42, 44, 50, 51, 53, 55]
reps = [1, 2, 5, 12, 16, 18, 19, 20, 21, 22, 28, 29, 30, 31, 37, 38, 39, 40,
        45, 46, 47, 48, 49, 54, 56]

df.loc[df['EST_ST'].isin(dems), 'polit'] = 'Dem'
df.loc[df['EST_ST'].isin(reps), 'polit'] = 'Rep'
df['polit'].value_counts()
```

Out[361…

```
Dem    45211
Rep    32614
Name: polit, dtype: int64
```

In [362…

```
# plot target means per state, color coded by democratic or republican vote
# for presidential candidate in the last election

with sns.plotting_context(context='poster'):
    fig, ax = plt.subplots(figsize=(30, 10))

    sns.pointplot(data=df, x='EST_ST', y='target', hue='polit',
                  ci=68, ax=ax, join=False, scale=1, capsize=0.5)
    ax.set_title("Target Mean per Category")
    ax.axhline(pop_mean, color='green', ls='dashed', label='population mean')
    ax.legend();
```



It does look like all of the states that cast the majority of their electoral college votes for Trump in the 2020 presidential election are at or above population mean, so they lean more towards vaccine hesitancy.

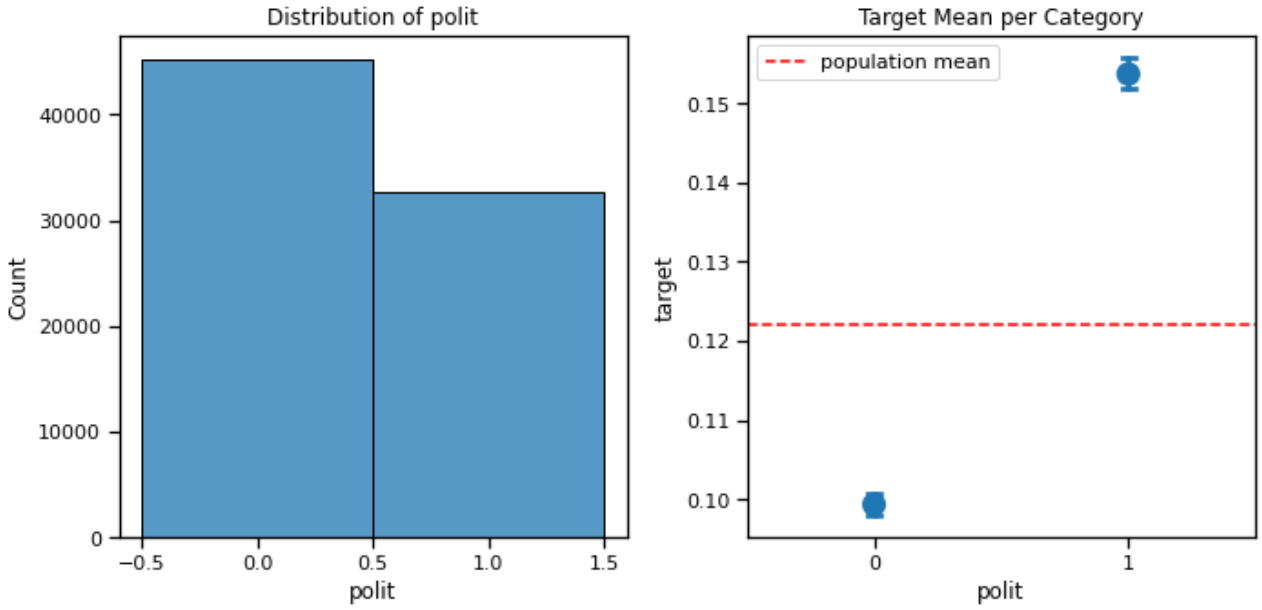
Some of the states that voted for Biden (orange) are at or above population mean too. But almost all of the states that are below population mean (lean more towards vaccine optimism) are Democratic majority.

Much as I wanted to avoid playing up the perception of a political divide, it does appear to be supported by the data. I will create a binary variable where 1 indicates the household was in a 'red' state, and 0 indicates a household in a 'blue' state.

In [411…
```
# Majority Democratic will be 0; majority Republican will be 1
df.loc[df['EST_ST'].isin(dems), 'polit'] = 0
df.loc[df['EST_ST'].isin(reps), 'polit'] = 1

cat_cols.append('polit')

df['polit'].value_counts()
```

Out[411…
```
0    45211
1    32614
Name: polit, dtype: int64
```

In [530…
```
# review polit's new distribution
dstools.explore_data_catbin(['polit'], df, 'target')
```



Plan to deal with missing values in preprocessing

I need to decide how each set of columns will be preprocessed. Categorical columns will be OHE and I can drop a label or not. Numeric columns will be standardized, but I need to decide how to deal with placeholders and impute values.

In [389…
```
# get a list of all the columns I intend to model with
all_cols = cat_cols + cat_cols_v1 + cat_cols_v2 + ord_cols + num_cols + \
           num_cols_v1 + num_cols_v2

print(len(all_cols))
print(len(set(all_cols)))
```

```
79
79
```

Categorical Columns

Some of my categorical columns have -2 values, which I will want to drop during OHE and will be replaced with the engineered 'incomplete' column. I need to separate these from the categorical columns which do NOT have -2 values, because I won't drop anything from those.

In [390…
```
# Create new dataframe including only the features I'll want to try out
# in my initial models
X = df[all_cols].copy()
y = df['target'].copy()

X.head()
```

Out[390…

	ANYWORK	CHNGHOW1	CHNGHOW10	CHNGHOW11	CHNGHOW12	CHNGHOW2	CHNGHOW3
0	0.0	1.0	0.0	0.0	0.0	1.0	0.0
1	1.0	1.0	0.0	0.0	0.0	1.0	0.0
2	1.0	-2.0	-2.0	-2.0	-2.0	-2.0	-2.0
3	1.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	1.0	0.0	1.0	0.0	1.0	0.0

5 rows × 79 columns

In [391…
```
# Loop through categorical columns and determine which have -2 and will need
# to have their first label dropped

all_cats = cat_cols + cat_cols_v1 + cat_cols_v2

cat_drop_first = []
cat_drop_none = []

for col in all_cats:
    if len(X.loc[X[col]==-2, col]) > 0:
        cat_drop_first.append(col)
    else:
        cat_drop_none.append(col)
```

In [392…
```
# Loop through lists to confirm results
for col in cat_drop_first:
    print(X[col].value_counts())
```

```
 0.0    38740
 1.0    34878
-2.0     4207
Name: CHNGHOW1, dtype: int64
 0.0    68799
 1.0     4819
-2.0     4207
Name: CHNGHOW10, dtype: int64
 0.0    72049
-2.0     4207
 1.0     1569
Name: CHNGHOW11, dtype: int64
 0.0    54509
 1.0    19109
-2.0     4207
Name: CHNGHOW12, dtype: int64
 0.0    56443
 1.0    17175
-2.0     4207
Name: CHNGHOW2, dtype: int64
 0.0    67959
 1.0     5659
```

```
-2.0     4207
Name: CHNGHOW3, dtype: int64
 0.0    51444
 1.0    22174
-2.0     4207
Name: CHNGHOW4, dtype: int64
 0.0    71127
-2.0     4207
 1.0     2491
Name: CHNGHOW5, dtype: int64
 0.0    36929
 1.0    36689
-2.0     4207
Name: CHNGHOW6, dtype: int64
 0.0    67489
 1.0     6129
-2.0     4207
Name: CHNGHOW7, dtype: int64
 0.0    58778
 1.0    14840
-2.0     4207
Name: CHNGHOW8, dtype: int64
 0.0    56697
 1.0    16921
-2.0     4207
Name: CHNGHOW9, dtype: int64
 0.0    46448
 1.0    17044
-2.0    13906
-1.0      427
Name: DELAY, dtype: int64
 4.0    49102
 3.0    12221
 1.0     7574
 2.0     6201
-2.0     2212
-1.0      515
Name: EIP, dtype: int64
 3.0    41270
 1.0    19747
 2.0     8917
-2.0     7438
-1.0      453
Name: FEWRTRANS, dtype: int64
 1.0    42247
 0.0    27499
-2.0     7438
-1.0      641
Name: FEWRTRIPS, dtype: int64
 0.0    64839
-2.0     8032
 1.0     4592
-1.0      362
Name: FREEFOOD, dtype: int64
 0.0    56873
-2.0    13906
 1.0     6658
-1.0      388
Name: MH_NOTGET, dtype: int64
 0.0    56552
-2.0    13906
 1.0     6911
-1.0      456
Name: MH_SVCS, dtype: int64
 0.0    51332
```

```
-2.0    13906
 1.0    12182
-1.0      405
Name: NOTGET, dtype: int64
 1.0    55601
 0.0    14386
-2.0     7438
-1.0      400
Name: PLNDTRIPS, dtype: int64
 0.0    49248
 1.0    14233
-2.0    13906
-1.0      438
Name: PRESCRIPT, dtype: int64
 0.0    64109
-2.0     8195
 1.0     4823
-1.0      698
Name: SNAP_YN, dtype: int64
 1.0    54736
 0.0    15651
-2.0     7438
Name: SPNDSRC1, dtype: int64
 0.0    52945
 1.0    17442
-2.0     7438
Name: SPNDSRC2, dtype: int64
 0.0    54435
 1.0    15952
-2.0     7438
Name: SPNDSRC3, dtype: int64
 0.0    65301
-2.0     7438
 1.0     5086
Name: SPNDSRC4, dtype: int64
 0.0    65345
-2.0     7438
 1.0     5042
Name: SPNDSRC5, dtype: int64
 0.0    57171
 1.0    13216
-2.0     7438
Name: SPNDSRC6, dtype: int64
 0.0    67820
-2.0     7438
 1.0     2567
Name: SPNDSRC7, dtype: int64
 0.0    67205
-2.0     7438
 1.0     3182
Name: SPNDSRC8, dtype: int64
 0.0    73011
 1.0     3343
-2.0      795
-1.0      676
Name: SSA_APPLY, dtype: int64
 0.0    51098
 1.0    25757
-2.0      519
-1.0      451
Name: SSA_RECV, dtype: int64
 2.0    30796
 1.0    17898
-2.0    14502
 3.0    13368
```

```
 4.0      812
-1.0      449
Name: TENURE, dtype: int64
 0.0    65536
 1.0    11875
-1.0      217
-2.0      197
Name: UI_APPLY, dtype: int64
 1.0    27320
 2.0    20314
-2.0    12614
 4.0    10040
 3.0     7173
-1.0      364
Name: ANXIOUS, dtype: int64
 1.0    34624
 2.0    19040
-2.0    12614
 4.0     5657
 3.0     5433
-1.0      457
Name: DOWN, dtype: int64
 1.0    32950
 2.0    19561
-2.0    12614
 3.0     6352
 4.0     5827
-1.0      521
Name: INTEREST, dtype: int64
 1.0    33379
 2.0    18586
-2.0    12614
 4.0     6921
 3.0     5847
-1.0      478
Name: WORRY, dtype: int64
 1.0    41607
 0.0    20467
-2.0    13204
-1.0     2547
Name: HLTHINS1, dtype: int64
 0.0    41859
 1.0    15103
-2.0    13204
-1.0     7659
Name: HLTHINS2, dtype: int64
 0.0    38522
 1.0    20068
-2.0    13204
-1.0     6031
Name: HLTHINS3, dtype: int64
 0.0    48768
-2.0    13204
-1.0     9221
 1.0     6632
Name: HLTHINS4, dtype: int64
 0.0    51916
-2.0    13204
-1.0     9702
 1.0     3003
Name: HLTHINS5, dtype: int64
 0.0    51515
-2.0    13204
-1.0    10056
 1.0     3050
```

```
Name: HLTHINS6, dtype: int64
 0.0    53368
-2.0    13204
-1.0    10689
 1.0      564
Name: HLTHINS7, dtype: int64
 0.0    48878
-2.0    13204
-1.0    12763
 1.0     2980
Name: HLTHINS8, dtype: int64
-2.0    57437
 1.0    14266
-1.0     6122
Name: ENROLL1, dtype: int64
-2.0    57437
-1.0    18627
 1.0     1761
Name: ENROLL2, dtype: int64
-2.0    57437
-1.0    15644
 1.0     4744
Name: ENROLL3, dtype: int64
 2.0     44955
-2.0     14502
 3.0      4975
 9.0      2730
 1.0      1921
 6.0      1746
 5.0      1729
 7.0      1562
 8.0      1479
 4.0      1254
-1.0       698
 10.0      274
Name: LIVQTR, dtype: int64
 1.0    61184
-2.0    13204
 0.0     3104
-1.0      333
Name: HLTHINS, dtype: int64
 0.0    57873
 1.0    14266
-2.0     5496
-1.0      190
Name: SCHOOL_KIDS, dtype: int64
 2.0    49930
-2.0    14502
 3.0    10500
 1.0     1921
-1.0      698
 4.0      274
Name: LIVQTR_2, dtype: int64
 0.0    45969
-2.0    15652
 1.0    14679
-1.0     1525
Name: TNUM_PS_2, dtype: int64
```

In [393…

```
# Loop through lists to confirm results
for col in cat_drop_none:
    print(X[col].value_counts())
```

```
 1.0    44266
 0.0    33312
```

```
-1.0      247
Name: ANYWORK, dtype: int64
0.0    46326
1.0    31499
Name: EGENDER, dtype: int64
 0.0    66184
 1.0    11349
-1.0      292
Name: EXPCTLOSS, dtype: int64
 0.0    68660
 1.0     8527
-1.0      638
Name: HADCOVID, dtype: int64
0.0    70428
1.0     7397
Name: RHISPANIC, dtype: int64
1.0    64253
2.0     6076
3.0     3925
4.0     3571
Name: RRACE, dtype: int64
 0.0    42009
 1.0    32097
-1.0     3719
Name: TW_START, dtype: int64
 0.0    48766
 1.0    28874
-1.0      185
Name: WRKLOSS, dtype: int64
2.0    43223
1.0    17308
3.0    17294
Name: THHLD_NUMADLT, dtype: int64
0.0        51865
47900.0     3050
35620.0     2111
37980.0     2071
14460.0     2020
42660.0     1848
41860.0     1842
31080.0     1751
19100.0     1680
16980.0     1679
26420.0     1632
38060.0     1429
40140.0     1229
19820.0     1222
12060.0     1220
33100.0     1176
Name: EST_MSA, dtype: int64
 1.0    45740
 5.0    13762
 3.0    11879
 2.0     4637
 4.0     1319
-1.0      488
Name: MS, dtype: int64
0.0    61514
1.0    16311
Name: inc_binary, dtype: int64
0.0    51865
1.0    25960
Name: IN_METRO_AREA, dtype: int64
 1.0    45740
 2.0    17835
```

```
 3.0    13762
-1.0      488
Name: MS_2, dtype: int64
```

Numeric and Ordinal columns

Now I have the numeric and ordinal columns which I will treat as numeric.

These will not be OHE, they will be standardized. I will need to determine the best way to remove the missing placeholders and impute them.

I think I will want to simply nullify and impute the -2 values, since their info will be covered by the 'incomplete' column. But for -1 values indicating someone skipped just that question, I will probably want to create another column to indicate the respondent skipped that particular question.

In [394…

```
# populate lists of numeric and ordinal columns that will not be OHE
# and which have placholders to be removed and imputed
# versus those which are fully filled
num_cols_filled = []
to_impute = []

for col in num_cols + num_cols_v1 + num_cols_v2 + ord_cols:
    vals = X[col].value_counts()
    if len(vals.loc[vals.index < 0]) > 0:
        to_impute.append(col)
    else:
        num_cols_filled.append(col)

print(to_impute)
print(num_cols_filled)
```

```
['PROP_FOODSPEND_HOME', 'TNUM_PS', 'CURFOODSUF', 'EXPNS_DIF', 'INCOME']
['THHLD_NUMKID', 'AGE', 'incomplete', 'skipped', 'EEDUC']
```

In [375…

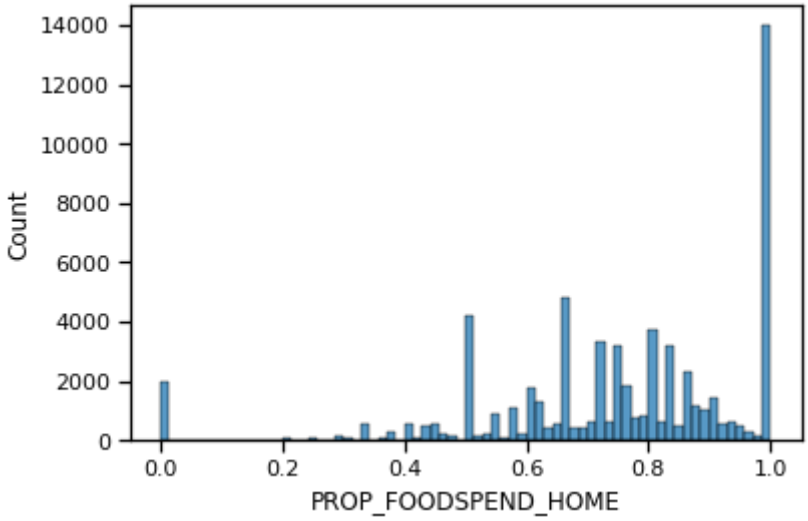
```
def missingind_prep(df, col, dummy_val=-1):
    """Prepares numeric columns that have placeholders for modeling.

    Create a separate missing indicator column where col is populated
    with `dummy_val`.

    Replace all values less than 0 with np.nan, since these numbers represent
    placeholders in this dataset.

    Graph the distribution to help me decide which statistic would be
    appropriate for imputing missing values.
    """
    skipped = col + '_skipped'

    # Create missing indicator column
    df.loc[df[col]==dummy_val, skipped] = 1
    df.loc[df[skipped].isna(), skipped] = 0

    # Nullify the -2 and -1 values to see the proportion distribution
    df.loc[df[col]<0, col] = np.nan

    # print descriptive statistics
    print(df[col].describe())
```

```
    #chart histogram
    fig, ax = plt.subplots()
    sns.histplot(df[col])
    plt.show();

    return df, skipped
```

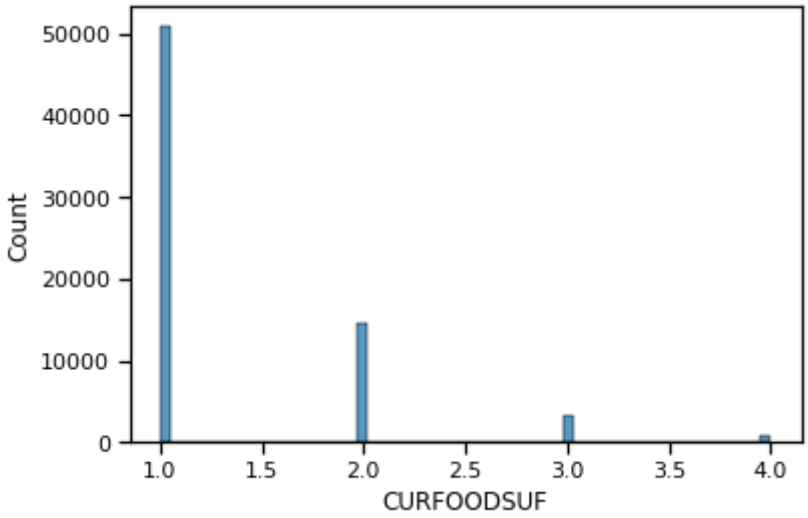
In [376…

```
missing_inds = []

for col in to_impute:
    # Create a missing indicator for -1 values
    X, ind_col = missingind_prep(X, col)
    missing_inds.append(ind_col)

missing_inds
```

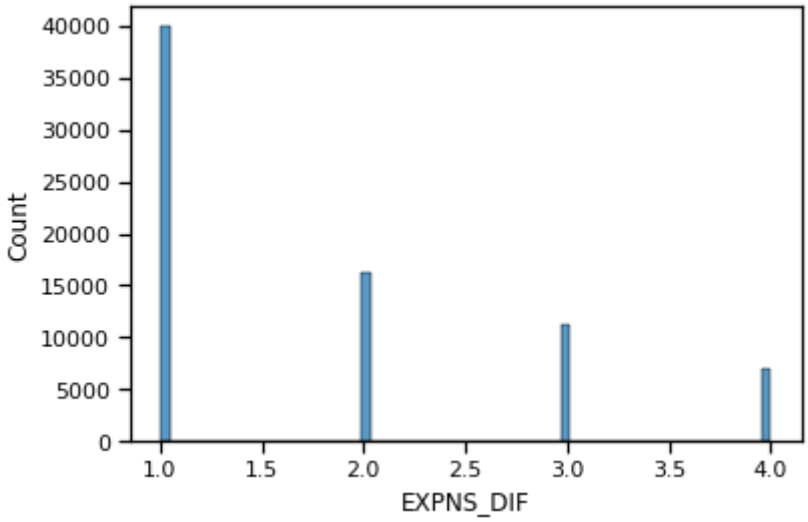
```
count    63810.000000
mean         0.745933
std          0.225950
min          0.000000
25%          0.636364
50%          0.769231
75%          0.923077
max          1.000000
Name: PROP_FOODSPEND_HOME, dtype: float64
```



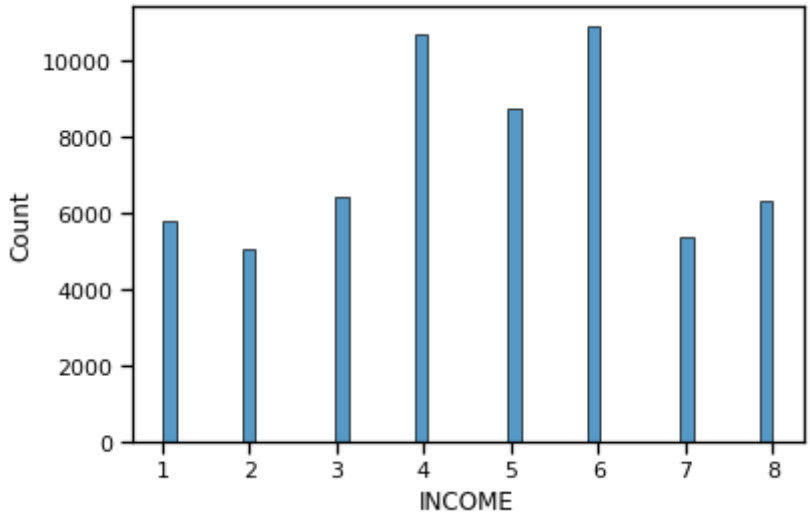

```
count    69644.000000
mean         1.340288
std          0.624428
min          1.000000
25%          1.000000
50%          1.000000
75%          2.000000
max          4.000000
Name: CURFOODSUF, dtype: float64
```



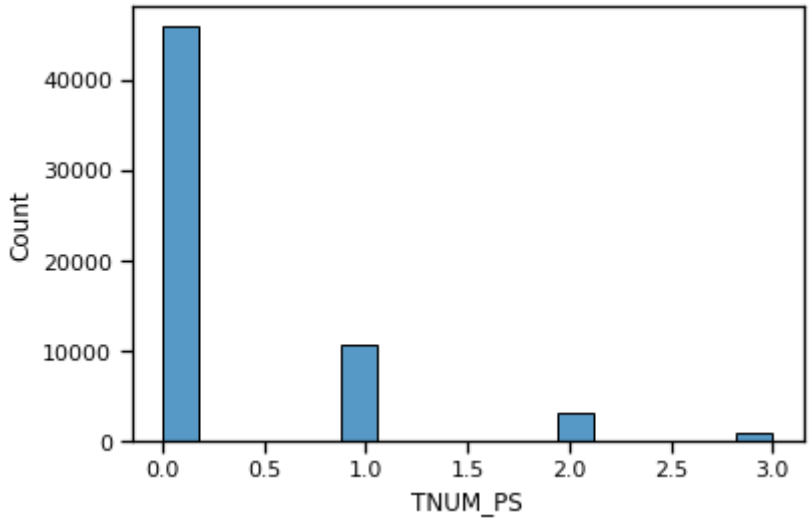

```
count    74463.000000
mean         1.801942
std          1.010840
min          1.000000
25%          1.000000
50%          1.000000
75%          2.000000
max          4.000000
Name: EXPNS_DIF, dtype: float64
```




```
count    59429.000000
mean         4.640916
std          2.079498
min          1.000000
25%          3.000000
50%          5.000000
75%          6.000000
max          8.000000
Name: INCOME, dtype: float64
```




```
count    60648.000000
mean         0.322237
std          0.638175
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          3.000000
Name: TNUM_PS, dtype: float64
```



Out[376…
```
['PROP_FOODSPEND_HOME_skipped',
 'CURFOODSUF_skipped',
 'EXPNS_DIF_skipped',
 'INCOME_skipped',
 'TNUM_PS_skipped']
```

PROP_FOODSPEND_HOME

- Represents the proportion of food spend that went towards food to be prepared at home versus food that already was prepared.
- By far the most common value is 1. Mean and median are around 0.75. I think the best bet is to impute this with mode.

TNUM_PS

- Represents the number of people in the household who plan to attend post-secondary classes.
- 0 is by far the most common, so will impute with mode.

CURFOODSUF

- Represents the level of food insufficiency in the past 7 days.
- 1 is by far the most common, so will impute with mode.

EXPNS_DIF

- Represents degree of difficulty meeting expenses
- Median and mode are both 1, but mean is significantly higher at 1.8 I think imputing with mean here is most representative.

INCOME

- Ordinally encoded income range pre-tax
- Mean seems like the most representative statistic to impute here.

In [395…

```
# append missing indicator columns to cat_col lists
cat_cols = cat_cols + missing_inds
cat_drop_none = cat_drop_none + missing_inds

num_cols_mode = ['PROP_FOODSPEND_HOME', 'TNUM_PS', 'CURFOODSUF']

num_cols_mean = ['EXPNS_DIF', 'INCOME']
```

Summary of Explore

During the Explore phase, I reviewed distributions of the original and engineered features. I also visualized the impact of each feature on the target, focusing on the target mean for categorical variables. Since target is binary, the mean of the target for each feature class is a representation of the proportion of vaccine hesitant respondents in that feature class.

There were a handful of features that didn't appear to have much impact on target mean, but at this point I'm not removing anything. I will model all of the features in their near-original forms first to get a baseline, and will iterate to remove features or substitute engineered versions of features to improve performance.

Going into the modeling phase, I have several lists of feature columns which have been split up based on how I intend to preprocess them, and whether they are the 'original' or more engineered features:

- **cat_drop_none**:
    - All categorical features, both original and engineered
    - Did not have any respondents who didn't answer because they prematurely exited the questionnaire. If -1 values exist indicating the respondent didn't answer that particular question only, these will be dummied in initial model.

- **cat_drop_first**:
    - All categorical features, both original and engineered
    - Had some respondents who didn't answer because they prematurely exited the questionnaire. These have value -2, so will drop first. If -1 values exist indicating the respondent didn't answer that particular question only, these will be dummied in initial model.
- **num_cols_mode**:
    - (almost) Original numeric features where values of -2 and -1 have been nullified
    - These will be scaled, since they're numeric
    - Null values to be imputed with mode
- **num_cols_mean**:
    - (almost) Original numeric features where values of -2 and -1 have been nullified
    - These will be scaled, since they're numeric
    - Null values to be imputed with mean
- **num_cols_filled**:
    - Numeric features I will include in my first model and that do not need to be imputed.
- **cat_cols**:
    - Categorical features that I didn't engineer anything additional for. These don't have a corresponding eng feature I can substitute.
- **cat_cols_v1**:
    - Original version of the categorical features that I did engineer. I will try these first, but am keeping a separate list so I can easily switch the engineered versions for these.
- **cat_cols_v2**:
    - Engineered versions of categorical features. These will not be included in initial model, but I'll transform them and keep them around so I can test them out if the original features don't perform too well.
- **num_cols_v1**:
    - Original versions of numeric features, which at this point is actually only the 'incomplete' feature. It will be replaced with 'inc_binary' in v2.
- **num_cols_v2**:
    - Engineered versions of numeric features, which at this point is actually only the 'skipped' feature. Depending on how the OHE -1 dummy columns and numeric "_skipped" columns perform, I can substitute those for this one.

MODEL

I will focus on F1_macro score, since I am modeling primarily for interpretation as opposed to prediction. I want the model to be as balanced as possible between precision and recall, and I

definitely want to make sure the model will be able to predict my target class of 1 - Hesitant even though it's imbalanced.

Because I have so many categorical features which will be dummied into one-hot columns, and the spread of my numeric features isn't too wide, I will use a Min-Max scaler instead of Standard scaler, because there will be more consistency between categorical and numeric that way.

In [186…
```
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.impute import SimpleImputer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegressionCV, LogisticRegression
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.svm import LinearSVC
from sklearn.ensemble import GradientBoostingClassifier, \
    RandomForestClassifier, BaggingClassifier
from sklearn.model_selection import train_test_split, cross_validate, \
    cross_val_predict, cross_val_score, GridSearchCV
from sklearn.feature_selection import SelectKBest, VarianceThreshold, chi2, \
    mutual_info_classif, f_classif, SelectFromModel
import sklearn.metrics as metrics
from sklearn.dummy import DummyClassifier
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.inspection import permutation_importance
from xgboost.sklearn import XGBClassifier
from xgboost import plot_importance
import joblib
```

Initial Preprocessing

Train-Test-Split

In [187…
```
# Perform train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
                                                    random_state=42)

X_train.head()
```

Out[187…

	ANYWORK	CHNGHOW1	CHNGHOW10	CHNGHOW11	CHNGHOW12	CHNGHOW2	CHNGH
29019	1.0	1.0	1.0	0.0	0.0	1.0	
5621	1.0	0.0	0.0	0.0	0.0	0.0	
67031	0.0	0.0	0.0	0.0	1.0	0.0	
1855	0.0	0.0	0.0	0.0	1.0	0.0	
4408	1.0	1.0	0.0	0.0	0.0	0.0	

5 rows × 84 columns

In [188…
```
print(y_train.value_counts(1))
y_train.value_counts()
```

```
0.0    0.878615
1.0    0.121385
```

```
Name: target, dtype: float64
```

Out[188…
```
0.0    51283
1.0     7085
Name: target, dtype: int64
```

In [189…
```
print(y_test.value_counts(1))
y_test.value_counts()
```

```
0.0    0.875623
1.0    0.124377
Name: target, dtype: float64
```

Out[189…
```
0.0    17037
1.0     2420
Name: target, dtype: int64
```

My target exhibits class imbalance.

The primary classification models I intend to test are:

- Decision Tree <-- has a class_weight parameter
- Random Forest <-- has a class_weight parameter

I will use SMOTE to resample my data and create a version of X_train that is resampled, so I can test its performance versus not adjusting and using class_weight=balanced.

Impute, OHE, Scale

Before I can resample to address class imbalance using SMOTE, I need to impute null values, since SMOTE doesn't like null values.

I may as well just build a column transformer of pipelines.

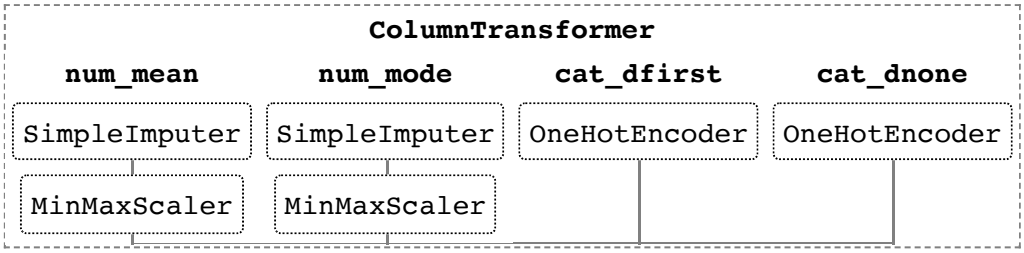
In [190…
```
# Pipeline for numeric columns imputing the mean
num_tf_mean = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', MinMaxScaler())
    ])

# Pipeline for numeric columns imputing the mode
num_tf_mode = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('scaler', MinMaxScaler())
    ])

# Pipeline for categorical columns where the first label will be dropped
cat_tf_dfirst = Pipeline(steps=[
    ('encoder', OneHotEncoder(sparse=False, drop='first'))
    ])

# Pipeline for categorical columns where the no label will be dropped
cat_tf_dnone = Pipeline(steps=[
    ('encoder', OneHotEncoder(sparse=False))
    ])

# create column transformer to do all the transformations

# Transforming and scaling all my X_train data, including some columns
# I may or may not end up using. This will avoid having to do it again later.
```

```
# Note that I'm adding num_cols_v2 and num_cols_filled with the mean
# transformer just to get them scaled somewhere;
# they have no missing values to be imputed

col_trans = ColumnTransformer(transformers=[
    ('num_mean', num_tf_mean, num_cols_mean + num_cols_v2 + num_cols_filled),
    ('num_mode', num_tf_mode, num_cols_mode),
    ('cat_dfirst', cat_tf_dfirst, cat_drop_first),
    ('cat_dnone', cat_tf_dnone, cat_drop_none)
    ])
```

In [191…

```
from sklearn import set_config
set_config(display='diagram')

col_trans
```

Out[191…



In [192…

```
# transform training data
X_train_tf = col_trans.fit_transform(X_train)
X_test_tf = col_trans.transform(X_test)
```

In [193…

```
# Do X_train and X_test have the same number of columns? They might not
# if I had rare columns in train but not test
print(X_train_tf.shape)
print(X_test_tf.shape)
```

```
(58368, 241)
(19457, 241)
```

In [194…

```
# Turn the X dfs into dataframes
nums = num_cols_mean + num_cols_v2 + num_cols_filled + num_cols_mode

# I'm assuming here that we had the same categories in X_test and X_train
ohe1 = list(col_trans.named_transformers_['cat_dfirst'].\
            named_steps['encoder'].get_feature_names(cat_drop_first))
ohe2 = list(col_trans.named_transformers_['cat_dnone'].\
            named_steps['encoder'].get_feature_names(cat_drop_none))

all_x_cols = nums + ohe1 + ohe2

X_train_df = pd.DataFrame(X_train_tf, columns=all_x_cols, index=X_train.index)
X_test_df = pd.DataFrame(X_test_tf, columns=all_x_cols, index=X_test.index)

display(X_train_df.head())
display(X_test_df.head())
```

	EXPNS_DIF	INCOME	skipped	THHLD_NUMKID	AGE	incomplete	EEDUC	PROP_
29019	0.000000	0.520189	0.000	0.4	0.385714	0.880952	1.000000	
5621	0.666667	0.714286	0.000	0.0	0.571429	0.000000	0.500000	
67031	0.000000	1.000000	0.175	0.0	0.700000	0.000000	1.000000	

	EXPNS_DIF	INCOME	skipped	THHLD_NUMKID	AGE	incomplete	EEDUC	PROP_
1855	0.000000	0.571429	0.025	0.2	0.614286	0.000000	0.666667	
4408	0.333333	0.571429	0.050	0.0	0.228571	0.000000	0.500000	

5 rows × 241 columns

	EXPNS_DIF	INCOME	skipped	THHLD_NUMKID	AGE	incomplete	EEDUC	PROP_
47933	0.333333	0.285714	0.05	0.0	0.714286	0.0	0.666667	
14435	0.000000	0.285714	0.05	0.0	0.842857	0.0	1.000000	
77494	1.000000	0.285714	0.00	0.2	0.414286	0.0	0.500000	
50202	0.000000	0.571429	0.00	0.0	0.542857	0.0	0.833333	
35105	0.333333	0.428571	0.00	0.0	0.685714	0.0	0.666667	

5 rows × 241 columns

In [195…
```
# How many -1 skipped question columns do I have now?
cols_skipped = list(filter(lambda x: x.endswith('-1.0'),
                           list(X_train_df.columns)))
cols_skipped
```

Out[195…
```
['DELAY_-1.0',
 'EIP_-1.0',
 'FEWRTRANS_-1.0',
 'FEWRTRIPS_-1.0',
 'FREEFOOD_-1.0',
 'MH_NOTGET_-1.0',
 'MH_SVCS_-1.0',
 'NOTGET_-1.0',
 'PLNDTRIPS_-1.0',
 'PRESCRIPT_-1.0',
 'SNAP_YN_-1.0',
 'SSA_APPLY_-1.0',
 'SSA_RECV_-1.0',
 'TENURE_-1.0',
 'UI_APPLY_-1.0',
 'ANXIOUS_-1.0',
 'DOWN_-1.0',
 'INTEREST_-1.0',
 'WORRY_-1.0',
 'HLTHINS1_-1.0',
 'HLTHINS2_-1.0',
 'HLTHINS3_-1.0',
 'HLTHINS4_-1.0',
 'HLTHINS5_-1.0',
 'HLTHINS6_-1.0',
 'HLTHINS7_-1.0',
 'HLTHINS8_-1.0',
 'ENROLL1_-1.0',
 'ENROLL2_-1.0',
 'ENROLL3_-1.0',
 'LIVQTR_-1.0',
 'HLTHINS_-1.0',
 'SCHOOL_KIDS_-1.0',
 'LIVQTR_2_-1.0',
```

```
 'TNUM_PS_2_-1.0',
 'ANYWORK_-1.0',
 'EXPCTLOSS_-1.0',
 'HADCOVID_-1.0',
 'TW_START_-1.0',
 'WRKLOSS_-1.0',
 'MS_-1.0',
 'MS_2_-1.0']
```

SMOTE

In [197…
```
from imblearn.over_sampling import SMOTENC
```

In [198…
```
# list columns to create a boolean mask for categorical
list(X_train_df.columns)
```

Out[198…
```
['EXPNS_DIF',
 'INCOME',
 'skipped',
 'THHLD_NUMKID',
 'AGE',
 'incomplete',
 'EEDUC',
 'PROP_FOODSPEND_HOME',
 'TNUM_PS',
 'CURFOODSUF',
 'CHNGHOW1_0.0',
 'CHNGHOW1_1.0',
 'CHNGHOW10_0.0',
 'CHNGHOW10_1.0',
 'CHNGHOW11_0.0',
 'CHNGHOW11_1.0',
 'CHNGHOW12_0.0',
 'CHNGHOW12_1.0',
 'CHNGHOW2_0.0',
 'CHNGHOW2_1.0',
 'CHNGHOW3_0.0',
 'CHNGHOW3_1.0',
 'CHNGHOW4_0.0',
 'CHNGHOW4_1.0',
 'CHNGHOW5_0.0',
 'CHNGHOW5_1.0',
 'CHNGHOW6_0.0',
 'CHNGHOW6_1.0',
 'CHNGHOW7_0.0',
 'CHNGHOW7_1.0',
 'CHNGHOW8_0.0',
 'CHNGHOW8_1.0',
 'CHNGHOW9_0.0',
 'CHNGHOW9_1.0',
 'DELAY_-1.0',
 'DELAY_0.0',
 'DELAY_1.0',
 'EIP_-1.0',
 'EIP_1.0',
 'EIP_2.0',
 'EIP_3.0',
 'EIP_4.0',
 'FEWRTRANS_-1.0',
 'FEWRTRANS_1.0',
 'FEWRTRANS_2.0',
 'FEWRTRANS_3.0',
 'FEWRTRIPS_-1.0',
```

```
'FEWRTRIPS_0.0',
'FEWRTRIPS_1.0',
'FREEFOOD_-1.0',
'FREEFOOD_0.0',
'FREEFOOD_1.0',
'MH_NOTGET_-1.0',
'MH_NOTGET_0.0',
'MH_NOTGET_1.0',
'MH_SVCS_-1.0',
'MH_SVCS_0.0',
'MH_SVCS_1.0',
'NOTGET_-1.0',
'NOTGET_0.0',
'NOTGET_1.0',
'PLNDTRIPS_-1.0',
'PLNDTRIPS_0.0',
'PLNDTRIPS_1.0',
'PRESCRIPT_-1.0',
'PRESCRIPT_0.0',
'PRESCRIPT_1.0',
'SNAP_YN_-1.0',
'SNAP_YN_0.0',
'SNAP_YN_1.0',
'SPNDSRC1_0.0',
'SPNDSRC1_1.0',
'SPNDSRC2_0.0',
'SPNDSRC2_1.0',
'SPNDSRC3_0.0',
'SPNDSRC3_1.0',
'SPNDSRC4_0.0',
'SPNDSRC4_1.0',
'SPNDSRC5_0.0',
'SPNDSRC5_1.0',
'SPNDSRC6_0.0',
'SPNDSRC6_1.0',
'SPNDSRC7_0.0',
'SPNDSRC7_1.0',
'SSA_APPLY_-1.0',
'SSA_APPLY_0.0',
'SSA_APPLY_1.0',
'SSA_RECV_-1.0',
'SSA_RECV_0.0',
'SSA_RECV_1.0',
'TENURE_-1.0',
'TENURE_1.0',
'TENURE_2.0',
'TENURE_3.0',
'TENURE_4.0',
'UI_APPLY_-1.0',
'UI_APPLY_0.0',
'UI_APPLY_1.0',
'ANXIOUS_-1.0',
'ANXIOUS_1.0',
'ANXIOUS_2.0',
'ANXIOUS_3.0',
'ANXIOUS_4.0',
'DOWN_-1.0',
'DOWN_1.0',
'DOWN_2.0',
'DOWN_3.0',
'DOWN_4.0',
'INTEREST_-1.0',
'INTEREST_1.0',
'INTEREST_2.0',
'INTEREST_3.0',
```

```
'INTEREST_4.0',
'WORRY_-1.0',
'WORRY_1.0',
'WORRY_2.0',
'WORRY_3.0',
'WORRY_4.0',
'HLTHINS1_-1.0',
'HLTHINS1_0.0',
'HLTHINS1_1.0',
'HLTHINS2_-1.0',
'HLTHINS2_0.0',
'HLTHINS2_1.0',
'HLTHINS3_-1.0',
'HLTHINS3_0.0',
'HLTHINS3_1.0',
'HLTHINS4_-1.0',
'HLTHINS4_0.0',
'HLTHINS4_1.0',
'HLTHINS5_-1.0',
'HLTHINS5_0.0',
'HLTHINS5_1.0',
'HLTHINS6_-1.0',
'HLTHINS6_0.0',
'HLTHINS6_1.0',
'HLTHINS7_-1.0',
'HLTHINS7_0.0',
'HLTHINS7_1.0',
'HLTHINS8_-1.0',
'HLTHINS8_0.0',
'HLTHINS8_1.0',
'ENROLL1_-1.0',
'ENROLL1_1.0',
'ENROLL2_-1.0',
'ENROLL2_1.0',
'ENROLL3_-1.0',
'ENROLL3_1.0',
'LIVQTR_-1.0',
'LIVQTR_1.0',
'LIVQTR_2.0',
'LIVQTR_3.0',
'LIVQTR_4.0',
'LIVQTR_5.0',
'LIVQTR_6.0',
'LIVQTR_7.0',
'LIVQTR_8.0',
'LIVQTR_9.0',
'LIVQTR_10.0',
'HLTHINS_-1.0',
'HLTHINS_0.0',
'HLTHINS_1.0',
'SCHOOL_KIDS_-1.0',
'SCHOOL_KIDS_0.0',
'SCHOOL_KIDS_1.0',
'LIVQTR_2_-1.0',
'LIVQTR_2_1.0',
'LIVQTR_2_2.0',
'LIVQTR_2_3.0',
'LIVQTR_2_4.0',
'TNUM_PS_2_-1.0',
'TNUM_PS_2_0.0',
'TNUM_PS_2_1.0',
'ANYWORK_-1.0',
'ANYWORK_0.0',
'ANYWORK_1.0',
'EGENDER_0.0',
```

```
'EGENDER_1.0',
'EXPCTLOSS_-1.0',
'EXPCTLOSS_0.0',
'EXPCTLOSS_1.0',
'HADCOVID_-1.0',
'HADCOVID_0.0',
'HADCOVID_1.0',
'RHISPANIC_0.0',
'RHISPANIC_1.0',
'RRACE_1.0',
'RRACE_2.0',
'RRACE_3.0',
'RRACE_4.0',
'TW_START_-1.0',
'TW_START_0.0',
'TW_START_1.0',
'WRKLOSS_-1.0',
'WRKLOSS_0.0',
'WRKLOSS_1.0',
'THHLD_NUMADLT_1.0',
'THHLD_NUMADLT_2.0',
'THHLD_NUMADLT_3.0',
'polit_0',
'polit_1',
'EST_MSA_0.0',
'EST_MSA_12060.0',
'EST_MSA_14460.0',
'EST_MSA_16980.0',
'EST_MSA_19100.0',
'EST_MSA_19820.0',
'EST_MSA_26420.0',
'EST_MSA_31080.0',
'EST_MSA_33100.0',
'EST_MSA_35620.0',
'EST_MSA_37980.0',
'EST_MSA_38060.0',
'EST_MSA_40140.0',
'EST_MSA_41860.0',
'EST_MSA_42660.0',
'EST_MSA_47900.0',
'MS_-1.0',
'MS_1.0',
'MS_2.0',
'MS_3.0',
'MS_4.0',
'MS_5.0',
'inc_binary_0.0',
'inc_binary_1.0',
'IN_METRO_AREA_0.0',
'IN_METRO_AREA_1.0',
'MS_2_-1.0',
'MS_2_1.0',
'MS_2_2.0',
'MS_2_3.0',
'PROP_FOODSPEND_HOME_skipped_0.0',
'PROP_FOODSPEND_HOME_skipped_1.0',
'TNUM_PS_skipped_0.0',
'TNUM_PS_skipped_1.0',
'CURFOODSUF_skipped_0.0',
'CURFOODSUF_skipped_1.0',
'EXPNS_DIF_skipped_0.0',
'EXPNS_DIF_skipped_1.0',
'INCOME_skipped_0.0',
'INCOME_skipped_1.0']
```

Luckily, the OHE columns were all put at the end, so I can easily make a boolean mask. The first 10 columns are False, and the rest are True.

In [199…

```
cat_mask = np.array([True]*10 + [False]*230)
print(len(cat_mask))
cat_mask
```

```
240
```

Out[199…

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False])
```

In [200…

```
# create SMOTEd versions of training data
sm = SMOTENC(cat_mask, random_state=42)
X_train_sm, y_train_sm = sm.fit_resample(X_train_df, y_train)
```

In [202…

```
# check to make sure class imbalance has been resolved
print(y_train_sm.value_counts(1))
print(y_train_sm.value_counts())
```

```
1.0    0.5
0.0    0.5
Name: target, dtype: float64
1.0    51283
0.0    51283
Name: target, dtype: int64
```

Group Features

I would like to create an initial model on the base (only one version) features and original / v1 features. This way I can compare the base+v1 model to the base+v2 features that are the versions I engineered.

I have different lists, but now that I've OHE many features, the lists aren't exact anymore.

I'll make a dataframe of columns in my training set and code which feature group each column should be part of.

Again, 'base' features are those that only have one version, so there isn't anything to trade out. Each v1 original column has at least one engineered v2 version I want to try instead.

Here is a summary of the main differences between v1 and v2 column sets:

Root column	V1	V2
HLTHINS	Multiple choice question consisting of 8 options that represented different types of health insurance coverage. Kept as separate columns which were OHE, including -1 for skipped.	Engineered into a single binary column representing whether the person has any health insurance coverage or not.
LIVQTR	Single choice question about housing, where there were separate choices for detached versus attached house, and apartment buildings with different numbers of apartments.	Combined both house choices and all apartment choices to consolidate them
TNUM_PS	Numeric column representing how many individuals in the household intended to attend post-secondary classes in 2021. Max reported was 3, most were 0.	Engineered into a single binary column representing whether anyone in the household planned to attend post-secondary classes.
EST_MSA	Categorical column populated with different values for the metropolitan area the household was in. Only top 10 metro areas were imputed at all.	Engineered into a single binary column representing whether the household was in a top 10 metro area or not.
ENROLL	Multiple choice question consisting of 3 options that represented whether the household contained children enrolled in school outside the home or homeschooled, or not enrolled in school. Kept as separate columns which were OHE, including -1 for skipped.	Engineered into single binary column representing whether the household contained any children who were attending school outside the home (homeschooled is 0)
MS	Single choice question about married status. Divorced, Widowed, and Separated were separate choices.	Combined Divorced, Widowed, and Separated into a single category, leaving Married, Never Married, and Was Married but aren't anymore
incomplete	Numeric column representing how many questions were left incomplete when the respondent prematurely exited the questionnaire. Value of 0 means they finished the whole thing.	Binary column indicating 1 for prematurely exited at any point, 0 for finished the whole thing.
skipped	Populated missing indicator columns for numeric, and OHE -1 values for categorical features to indicate which specific questions a respondent didn't answer. Each question has its oen skipped indicator in some form in V1. This is different from incomplete, which indicates the entire questionnaire was exited at a certain point; skipped just means they skipped a particular question and kept going.	Numeric column with the number of questions the respondent skipped. This replaces all the individual skipped indicator columns.

In [582…

```
# create a dataframe with the preprocessed column names as the index and
# a single column filled with nans
df_feats = pd.DataFrame(np.full([len(X_train_df.columns), 2], np.nan),
                        columns=['col_root', 'feat_group'],
                        index=list(X_train_df.columns))

# Where index doesn't end with '.0', the column is NOT OHE and col_root
# should be index
find = ~df_feats.index.str.endswith('.0')
df_feats.loc[find, 'col_root'] = df_feats[find].index

# what ends with '.0' is OHE and I need to take off the label part
find = df_feats.index.str.endswith('.0')

# split off the last bit after an underscore, knowing some roots have _ too
roots = ['_'.join(x.split('_')[:-1]) for x in df_feats[find].index]
# also split off the 'skipped'
roots = [x.replace('_skipped', '') for x in roots]

# update roots for OHE columns into df
df_feats.loc[find, 'col_root'] = roots

# Update polit
find = df_feats.index.str.startswith('polit_')
df_feats.loc[find, 'col_root'] = 'polit'

with pd.option_context('display.max_rows', 250):
    display(df_feats)
```

	col_root	feat_group
EXPNS_DIF	EXPNS_DIF	NaN
INCOME	INCOME	NaN
skipped	skipped	NaN
THHLD_NUMKID	THHLD_NUMKID	NaN
AGE	AGE	NaN
incomplete	incomplete	NaN
EEDUC	EEDUC	NaN
PROP_FOODSPEND_HOME	PROP_FOODSPEND_HOME	NaN
TNUM_PS	TNUM_PS	NaN
CURFOODSUF	CURFOODSUF	NaN
CHNGHOW1_0.0	CHNGHOW1	NaN
CHNGHOW1_1.0	CHNGHOW1	NaN
CHNGHOW10_0.0	CHNGHOW10	NaN
CHNGHOW10_1.0	CHNGHOW10	NaN
CHNGHOW11_0.0	CHNGHOW11	NaN
CHNGHOW11_1.0	CHNGHOW11	NaN
CHNGHOW12_0.0	CHNGHOW12	NaN

	col_root	feat_group
CHNGHOW12_1.0	CHNGHOW12	NaN
CHNGHOW2_0.0	CHNGHOW2	NaN
CHNGHOW2_1.0	CHNGHOW2	NaN
CHNGHOW3_0.0	CHNGHOW3	NaN
CHNGHOW3_1.0	CHNGHOW3	NaN
CHNGHOW4_0.0	CHNGHOW4	NaN
CHNGHOW4_1.0	CHNGHOW4	NaN
CHNGHOW5_0.0	CHNGHOW5	NaN
CHNGHOW5_1.0	CHNGHOW5	NaN
CHNGHOW6_0.0	CHNGHOW6	NaN
CHNGHOW6_1.0	CHNGHOW6	NaN
CHNGHOW7_0.0	CHNGHOW7	NaN
CHNGHOW7_1.0	CHNGHOW7	NaN
CHNGHOW8_0.0	CHNGHOW8	NaN
CHNGHOW8_1.0	CHNGHOW8	NaN
CHNGHOW9_0.0	CHNGHOW9	NaN
CHNGHOW9_1.0	CHNGHOW9	NaN
DELAY_-1.0	DELAY	NaN
DELAY_0.0	DELAY	NaN
DELAY_1.0	DELAY	NaN
EIP_-1.0	EIP	NaN
EIP_1.0	EIP	NaN
EIP_2.0	EIP	NaN
EIP_3.0	EIP	NaN
EIP_4.0	EIP	NaN
FEWRTRANS_-1.0	FEWRTRANS	NaN
FEWRTRANS_1.0	FEWRTRANS	NaN
FEWRTRANS_2.0	FEWRTRANS	NaN
FEWRTRANS_3.0	FEWRTRANS	NaN
FEWRTRIPS_-1.0	FEWRTRIPS	NaN
FEWRTRIPS_0.0	FEWRTRIPS	NaN
FEWRTRIPS_1.0	FEWRTRIPS	NaN
FREEFOOD_-1.0	FREEFOOD	NaN
FREEFOOD_0.0	FREEFOOD	NaN
FREEFOOD_1.0	FREEFOOD	NaN

	col_root	feat_group
MH_NOTGET_-1.0	MH_NOTGET	NaN
MH_NOTGET_0.0	MH_NOTGET	NaN
MH_NOTGET_1.0	MH_NOTGET	NaN
MH_SVCS_-1.0	MH_SVCS	NaN
MH_SVCS_0.0	MH_SVCS	NaN
MH_SVCS_1.0	MH_SVCS	NaN
NOTGET_-1.0	NOTGET	NaN
NOTGET_0.0	NOTGET	NaN
NOTGET_1.0	NOTGET	NaN
PLNDTRIPS_-1.0	PLNDTRIPS	NaN
PLNDTRIPS_0.0	PLNDTRIPS	NaN
PLNDTRIPS_1.0	PLNDTRIPS	NaN
PRESCRIPT_-1.0	PRESCRIPT	NaN
PRESCRIPT_0.0	PRESCRIPT	NaN
PRESCRIPT_1.0	PRESCRIPT	NaN
SNAP_YN_-1.0	SNAP_YN	NaN
SNAP_YN_0.0	SNAP_YN	NaN
SNAP_YN_1.0	SNAP_YN	NaN
SPNDSRC1_0.0	SPNDSRC1	NaN
SPNDSRC1_1.0	SPNDSRC1	NaN
SPNDSRC2_0.0	SPNDSRC2	NaN
SPNDSRC2_1.0	SPNDSRC2	NaN
SPNDSRC3_0.0	SPNDSRC3	NaN
SPNDSRC3_1.0	SPNDSRC3	NaN
SPNDSRC4_0.0	SPNDSRC4	NaN
SPNDSRC4_1.0	SPNDSRC4	NaN
SPNDSRC5_0.0	SPNDSRC5	NaN
SPNDSRC5_1.0	SPNDSRC5	NaN
SPNDSRC6_0.0	SPNDSRC6	NaN
SPNDSRC6_1.0	SPNDSRC6	NaN
SPNDSRC7_0.0	SPNDSRC7	NaN
SPNDSRC7_1.0	SPNDSRC7	NaN
SSA_APPLY_-1.0	SSA_APPLY	NaN
SSA_APPLY_0.0	SSA_APPLY	NaN
SSA_APPLY_1.0	SSA_APPLY	NaN

	col_root	feat_group
SSA_RECV_-1.0	SSA_RECV	NaN
SSA_RECV_0.0	SSA_RECV	NaN
SSA_RECV_1.0	SSA_RECV	NaN
TENURE_-1.0	TENURE	NaN
TENURE_1.0	TENURE	NaN
TENURE_2.0	TENURE	NaN
TENURE_3.0	TENURE	NaN
TENURE_4.0	TENURE	NaN
UI_APPLY_-1.0	UI_APPLY	NaN
UI_APPLY_0.0	UI_APPLY	NaN
UI_APPLY_1.0	UI_APPLY	NaN
ANXIOUS_-1.0	ANXIOUS	NaN
ANXIOUS_1.0	ANXIOUS	NaN
ANXIOUS_2.0	ANXIOUS	NaN
ANXIOUS_3.0	ANXIOUS	NaN
ANXIOUS_4.0	ANXIOUS	NaN
DOWN_-1.0	DOWN	NaN
DOWN_1.0	DOWN	NaN
DOWN_2.0	DOWN	NaN
DOWN_3.0	DOWN	NaN
DOWN_4.0	DOWN	NaN
INTEREST_-1.0	INTEREST	NaN
INTEREST_1.0	INTEREST	NaN
INTEREST_2.0	INTEREST	NaN
INTEREST_3.0	INTEREST	NaN
INTEREST_4.0	INTEREST	NaN
WORRY_-1.0	WORRY	NaN
WORRY_1.0	WORRY	NaN
WORRY_2.0	WORRY	NaN
WORRY_3.0	WORRY	NaN
WORRY_4.0	WORRY	NaN
HLTHINS1_-1.0	HLTHINS1	NaN
HLTHINS1_0.0	HLTHINS1	NaN
HLTHINS1_1.0	HLTHINS1	NaN
HLTHINS2_-1.0	HLTHINS2	NaN

	col_root	feat_group
HLTHINS2_0.0	HLTHINS2	NaN
HLTHINS2_1.0	HLTHINS2	NaN
HLTHINS3_-1.0	HLTHINS3	NaN
HLTHINS3_0.0	HLTHINS3	NaN
HLTHINS3_1.0	HLTHINS3	NaN
HLTHINS4_-1.0	HLTHINS4	NaN
HLTHINS4_0.0	HLTHINS4	NaN
HLTHINS4_1.0	HLTHINS4	NaN
HLTHINS5_-1.0	HLTHINS5	NaN
HLTHINS5_0.0	HLTHINS5	NaN
HLTHINS5_1.0	HLTHINS5	NaN
HLTHINS6_-1.0	HLTHINS6	NaN
HLTHINS6_0.0	HLTHINS6	NaN
HLTHINS6_1.0	HLTHINS6	NaN
HLTHINS7_-1.0	HLTHINS7	NaN
HLTHINS7_0.0	HLTHINS7	NaN
HLTHINS7_1.0	HLTHINS7	NaN
HLTHINS8_-1.0	HLTHINS8	NaN
HLTHINS8_0.0	HLTHINS8	NaN
HLTHINS8_1.0	HLTHINS8	NaN
ENROLL1_-1.0	ENROLL1	NaN
ENROLL1_1.0	ENROLL1	NaN
ENROLL2_-1.0	ENROLL2	NaN
ENROLL2_1.0	ENROLL2	NaN
ENROLL3_-1.0	ENROLL3	NaN
ENROLL3_1.0	ENROLL3	NaN
LIVQTR_-1.0	LIVQTR	NaN
LIVQTR_1.0	LIVQTR	NaN
LIVQTR_2.0	LIVQTR	NaN
LIVQTR_3.0	LIVQTR	NaN
LIVQTR_4.0	LIVQTR	NaN
LIVQTR_5.0	LIVQTR	NaN
LIVQTR_6.0	LIVQTR	NaN
LIVQTR_7.0	LIVQTR	NaN
LIVQTR_8.0	LIVQTR	NaN

	col_root	feat_group
LIVQTR_9.0	LIVQTR	NaN
LIVQTR_10.0	LIVQTR	NaN
HLTHINS_-1.0	HLTHINS	NaN
HLTHINS_0.0	HLTHINS	NaN
HLTHINS_1.0	HLTHINS	NaN
SCHOOL_KIDS_-1.0	SCHOOL_KIDS	NaN
SCHOOL_KIDS_0.0	SCHOOL_KIDS	NaN
SCHOOL_KIDS_1.0	SCHOOL_KIDS	NaN
LIVQTR_2_-1.0	LIVQTR_2	NaN
LIVQTR_2_1.0	LIVQTR_2	NaN
LIVQTR_2_2.0	LIVQTR_2	NaN
LIVQTR_2_3.0	LIVQTR_2	NaN
LIVQTR_2_4.0	LIVQTR_2	NaN
TNUM_PS_2_-1.0	TNUM_PS_2	NaN
TNUM_PS_2_0.0	TNUM_PS_2	NaN
TNUM_PS_2_1.0	TNUM_PS_2	NaN
ANYWORK_-1.0	ANYWORK	NaN
ANYWORK_0.0	ANYWORK	NaN
ANYWORK_1.0	ANYWORK	NaN
EGENDER_0.0	EGENDER	NaN
EGENDER_1.0	EGENDER	NaN
EXPCTLOSS_-1.0	EXPCTLOSS	NaN
EXPCTLOSS_0.0	EXPCTLOSS	NaN
EXPCTLOSS_1.0	EXPCTLOSS	NaN
HADCOVID_-1.0	HADCOVID	NaN
HADCOVID_0.0	HADCOVID	NaN
HADCOVID_1.0	HADCOVID	NaN
RHISPANIC_0.0	RHISPANIC	NaN
RHISPANIC_1.0	RHISPANIC	NaN
RRACE_1.0	RRACE	NaN
RRACE_2.0	RRACE	NaN
RRACE_3.0	RRACE	NaN
RRACE_4.0	RRACE	NaN
TW_START_-1.0	TW_START	NaN
TW_START_0.0	TW_START	NaN

	col_root	feat_group
TW_START_1.0	TW_START	NaN
WRKLOSS_-1.0	WRKLOSS	NaN
WRKLOSS_0.0	WRKLOSS	NaN
WRKLOSS_1.0	WRKLOSS	NaN
THHLD_NUMADLT_1.0	THHLD_NUMADLT	NaN
THHLD_NUMADLT_2.0	THHLD_NUMADLT	NaN
THHLD_NUMADLT_3.0	THHLD_NUMADLT	NaN
polit_0	polit	NaN
polit_1	polit	NaN
EST_MSA_0.0	EST_MSA	NaN
EST_MSA_12060.0	EST_MSA	NaN
EST_MSA_14460.0	EST_MSA	NaN
EST_MSA_16980.0	EST_MSA	NaN
EST_MSA_19100.0	EST_MSA	NaN
EST_MSA_19820.0	EST_MSA	NaN
EST_MSA_26420.0	EST_MSA	NaN
EST_MSA_31080.0	EST_MSA	NaN
EST_MSA_33100.0	EST_MSA	NaN
EST_MSA_35620.0	EST_MSA	NaN
EST_MSA_37980.0	EST_MSA	NaN
EST_MSA_38060.0	EST_MSA	NaN
EST_MSA_40140.0	EST_MSA	NaN
EST_MSA_41860.0	EST_MSA	NaN
EST_MSA_42660.0	EST_MSA	NaN
EST_MSA_47900.0	EST_MSA	NaN
MS_-1.0	MS	NaN
MS_1.0	MS	NaN
MS_2.0	MS	NaN
MS_3.0	MS	NaN
MS_4.0	MS	NaN
MS_5.0	MS	NaN
inc_binary_0.0	inc_binary	NaN
inc_binary_1.0	inc_binary	NaN
IN_METRO_AREA_0.0	IN_METRO_AREA	NaN
IN_METRO_AREA_1.0	IN_METRO_AREA	NaN

	col_root	feat_group
MS_2_-1.0	MS_2	NaN
MS_2_1.0	MS_2	NaN
MS_2_2.0	MS_2	NaN
MS_2_3.0	MS_2	NaN
PROP_FOODSPEND_HOME_skipped_0.0	PROP_FOODSPEND_HOME	NaN
PROP_FOODSPEND_HOME_skipped_1.0	PROP_FOODSPEND_HOME	NaN
TNUM_PS_skipped_0.0	TNUM_PS	NaN
TNUM_PS_skipped_1.0	TNUM_PS	NaN
CURFOODSUF_skipped_0.0	CURFOODSUF	NaN
CURFOODSUF_skipped_1.0	CURFOODSUF	NaN
EXPNS_DIF_skipped_0.0	EXPNS_DIF	NaN
EXPNS_DIF_skipped_1.0	EXPNS_DIF	NaN
INCOME_skipped_0.0	INCOME	NaN
INCOME_skipped_1.0	INCOME	NaN

In [583…

```
# fill in feat_group based on root column from lists
base = num_cols_mode + num_cols_mean + num_cols_filled + cat_cols
v1 = cat_cols_v1 + num_cols_v1
v2 = num_cols_v2 + cat_cols_v2

df_feats.loc[df_feats['col_root'].isin(base), 'feat_group'] = 'base'
df_feats.loc[df_feats['col_root'].isin(v1), 'feat_group'] = 'v1'
df_feats.loc[df_feats['col_root'].isin(v2), 'feat_group'] = 'v2'

# I will include the _-1.0 and _skipped cols in v1 but not v2, since v2 has a
# single 'skipped' column for any skipped questions
# need to change those columns from feat_group = base to v1
find = df_feats.index.str.endswith('_-1.0')
df_feats.loc[(find) & (df_feats['feat_group']=='base'), 'feat_group'] = 'v1'
find = df_feats.index.str.endswith('_skipped_1.0')
df_feats.loc[(find) & (df_feats['feat_group']=='base'), 'feat_group'] = 'v1'

# _-1.0 skipped columns for v2 engineered features will be replaced with the
# general skipped column, so can be removed
find = df_feats.index.str.endswith('_-1.0')
df_feats.loc[(find) & (df_feats['feat_group']=='v2'), 'feat_group'] = 'removed'

# The numeric columns have skipped_0.0 which I don't need in any sets
# since I have their actual answers instead
find = df_feats.index.str.endswith('_skipped_0.0')
df_feats.loc[find, 'feat_group'] = 'removed'

df_feats.sort_index(axis=0, inplace=True)

df_feats.head()
```

Out[583…

	col_root	feat_group
AGE	AGE	base

	col_root	feat_group
ANXIOUS_-1.0	ANXIOUS	v1
ANXIOUS_1.0	ANXIOUS	base
ANXIOUS_2.0	ANXIOUS	base
ANXIOUS_3.0	ANXIOUS	base

In [584…

```
# Drop Category 0 or 1 from OHE variables
# I'm not going to drop -1 because they are so rare. 0 or 1 is the best option

# filter out the _-1.0 skipped because I don't want to drop those as reference
no_skipped = ~df_feats.index.str.endswith('_-1.0')

# filter out a list of columns that won't need to have a cat dropped
# or need to be dealt with manually
manual = ['CURFOODSUF','ENROLL1','ENROLL2','ENROLL3','EXPNS_DIF','INCOME',
          'LIVQTR','LIVQTR_2','PROP_FOODSPEND_HOME','TNUM_PS']
no_manual = ~df_feats['col_root'].isin(manual)

# filter for only columns that are OHE (root is listed more than once)
roots = df_feats.groupby('col_root')['col_root'].count()
ohe_roots = df_feats['col_root'].isin(list(roots[roots > 1].index))

# group by root, to get count of dummy columns in each category
# -1.0 skipped dummy columns filtered out already
dummies = df_feats.loc[(df_feats['feat_group'].isin(['base', 'v1', 'v2']))
             & (no_skipped) & (ohe_roots) & (no_manual), 'col_root']

dummies
```

Out[584…

```
ANXIOUS_1.0          ANXIOUS
ANXIOUS_2.0          ANXIOUS
ANXIOUS_3.0          ANXIOUS
ANXIOUS_4.0          ANXIOUS
ANYWORK_0.0          ANYWORK
                     ...
WRKLOSS_1.0          WRKLOSS
inc_binary_0.0    inc_binary
inc_binary_1.0    inc_binary
polit_0               polit
polit_1               polit
Name: col_root, Length: 162, dtype: object
```

In [585…

```
# loop through and mark first remaining non-1 column for each root to be dropped
for root in list(set(dummies)):
    fam = dummies.loc[df_feats['col_root']==root].sort_index(axis=0)
    first_ix = fam.index[0]
    df_feats.loc[df_feats.index == first_ix, 'feat_group'] = 'dummy_drop'


with pd.option_context('display.max_rows', 250):
    display(df_feats)
```

	col_root	feat_group
AGE	AGE	base
ANXIOUS_-1.0	ANXIOUS	v1

	col_root	feat_group
ANXIOUS_1.0	ANXIOUS	dummy_drop
ANXIOUS_2.0	ANXIOUS	base
ANXIOUS_3.0	ANXIOUS	base
ANXIOUS_4.0	ANXIOUS	base
ANYWORK_-1.0	ANYWORK	v1
ANYWORK_0.0	ANYWORK	dummy_drop
ANYWORK_1.0	ANYWORK	base
CHNGHOW10_0.0	CHNGHOW10	dummy_drop
CHNGHOW10_1.0	CHNGHOW10	base
CHNGHOW11_0.0	CHNGHOW11	dummy_drop
CHNGHOW11_1.0	CHNGHOW11	base
CHNGHOW12_0.0	CHNGHOW12	dummy_drop
CHNGHOW12_1.0	CHNGHOW12	base
CHNGHOW1_0.0	CHNGHOW1	dummy_drop
CHNGHOW1_1.0	CHNGHOW1	base
CHNGHOW2_0.0	CHNGHOW2	dummy_drop
CHNGHOW2_1.0	CHNGHOW2	base
CHNGHOW3_0.0	CHNGHOW3	dummy_drop
CHNGHOW3_1.0	CHNGHOW3	base
CHNGHOW4_0.0	CHNGHOW4	dummy_drop
CHNGHOW4_1.0	CHNGHOW4	base
CHNGHOW5_0.0	CHNGHOW5	dummy_drop
CHNGHOW5_1.0	CHNGHOW5	base
CHNGHOW6_0.0	CHNGHOW6	dummy_drop
CHNGHOW6_1.0	CHNGHOW6	base
CHNGHOW7_0.0	CHNGHOW7	dummy_drop
CHNGHOW7_1.0	CHNGHOW7	base
CHNGHOW8_0.0	CHNGHOW8	dummy_drop
CHNGHOW8_1.0	CHNGHOW8	base
CHNGHOW9_0.0	CHNGHOW9	dummy_drop
CHNGHOW9_1.0	CHNGHOW9	base
CURFOODSUF	CURFOODSUF	base
CURFOODSUF_skipped_0.0	CURFOODSUF	removed
CURFOODSUF_skipped_1.0	CURFOODSUF	v1
DELAY_-1.0	DELAY	v1

	col_root	feat_group
DELAY_0.0	DELAY	dummy_drop
DELAY_1.0	DELAY	base
DOWN_-1.0	DOWN	v1
DOWN_1.0	DOWN	dummy_drop
DOWN_2.0	DOWN	base
DOWN_3.0	DOWN	base
DOWN_4.0	DOWN	base
EEDUC	EEDUC	base
EGENDER_0.0	EGENDER	dummy_drop
EGENDER_1.0	EGENDER	base
EIP_-1.0	EIP	v1
EIP_1.0	EIP	dummy_drop
EIP_2.0	EIP	base
EIP_3.0	EIP	base
EIP_4.0	EIP	base
ENROLL1_-1.0	ENROLL1	v1
ENROLL1_1.0	ENROLL1	v1
ENROLL2_-1.0	ENROLL2	v1
ENROLL2_1.0	ENROLL2	v1
ENROLL3_-1.0	ENROLL3	v1
ENROLL3_1.0	ENROLL3	v1
EST_MSA_0.0	EST_MSA	dummy_drop
EST_MSA_12060.0	EST_MSA	v1
EST_MSA_14460.0	EST_MSA	v1
EST_MSA_16980.0	EST_MSA	v1
EST_MSA_19100.0	EST_MSA	v1
EST_MSA_19820.0	EST_MSA	v1
EST_MSA_26420.0	EST_MSA	v1
EST_MSA_31080.0	EST_MSA	v1
EST_MSA_33100.0	EST_MSA	v1
EST_MSA_35620.0	EST_MSA	v1
EST_MSA_37980.0	EST_MSA	v1
EST_MSA_38060.0	EST_MSA	v1
EST_MSA_40140.0	EST_MSA	v1
EST_MSA_41860.0	EST_MSA	v1

	col_root	feat_group
EST_MSA_42660.0	EST_MSA	v1
EST_MSA_47900.0	EST_MSA	v1
EXPCTLOSS_-1.0	EXPCTLOSS	v1
EXPCTLOSS_0.0	EXPCTLOSS	dummy_drop
EXPCTLOSS_1.0	EXPCTLOSS	base
EXPNS_DIF	EXPNS_DIF	base
EXPNS_DIF_skipped_0.0	EXPNS_DIF	removed
EXPNS_DIF_skipped_1.0	EXPNS_DIF	v1
FEWRTRANS_-1.0	FEWRTRANS	v1
FEWRTRANS_1.0	FEWRTRANS	dummy_drop
FEWRTRANS_2.0	FEWRTRANS	base
FEWRTRANS_3.0	FEWRTRANS	base
FEWRTRIPS_-1.0	FEWRTRIPS	v1
FEWRTRIPS_0.0	FEWRTRIPS	dummy_drop
FEWRTRIPS_1.0	FEWRTRIPS	base
FREEFOOD_-1.0	FREEFOOD	v1
FREEFOOD_0.0	FREEFOOD	dummy_drop
FREEFOOD_1.0	FREEFOOD	base
HADCOVID_-1.0	HADCOVID	v1
HADCOVID_0.0	HADCOVID	dummy_drop
HADCOVID_1.0	HADCOVID	base
HLTHINS1_-1.0	HLTHINS1	v1
HLTHINS1_0.0	HLTHINS1	dummy_drop
HLTHINS1_1.0	HLTHINS1	v1
HLTHINS2_-1.0	HLTHINS2	v1
HLTHINS2_0.0	HLTHINS2	dummy_drop
HLTHINS2_1.0	HLTHINS2	v1
HLTHINS3_-1.0	HLTHINS3	v1
HLTHINS3_0.0	HLTHINS3	dummy_drop
HLTHINS3_1.0	HLTHINS3	v1
HLTHINS4_-1.0	HLTHINS4	v1
HLTHINS4_0.0	HLTHINS4	dummy_drop
HLTHINS4_1.0	HLTHINS4	v1
HLTHINS5_-1.0	HLTHINS5	v1
HLTHINS5_0.0	HLTHINS5	dummy_drop

	col_root	feat_group
HLTHINS5_1.0	HLTHINS5	v1
HLTHINS6_-1.0	HLTHINS6	v1
HLTHINS6_0.0	HLTHINS6	dummy_drop
HLTHINS6_1.0	HLTHINS6	v1
HLTHINS7_-1.0	HLTHINS7	v1
HLTHINS7_0.0	HLTHINS7	dummy_drop
HLTHINS7_1.0	HLTHINS7	v1
HLTHINS8_-1.0	HLTHINS8	v1
HLTHINS8_0.0	HLTHINS8	dummy_drop
HLTHINS8_1.0	HLTHINS8	v1
HLTHINS_-1.0	HLTHINS	removed
HLTHINS_0.0	HLTHINS	dummy_drop
HLTHINS_1.0	HLTHINS	v2
INCOME	INCOME	base
INCOME_skipped_0.0	INCOME	removed
INCOME_skipped_1.0	INCOME	v1
INTEREST_-1.0	INTEREST	v1
INTEREST_1.0	INTEREST	dummy_drop
INTEREST_2.0	INTEREST	base
INTEREST_3.0	INTEREST	base
INTEREST_4.0	INTEREST	base
IN_METRO_AREA_0.0	IN_METRO_AREA	dummy_drop
IN_METRO_AREA_1.0	IN_METRO_AREA	v2
LIVQTR_-1.0	LIVQTR	v1
LIVQTR_1.0	LIVQTR	v1
LIVQTR_10.0	LIVQTR	v1
LIVQTR_2.0	LIVQTR	v1
LIVQTR_2_-1.0	LIVQTR_2	removed
LIVQTR_2_1.0	LIVQTR_2	v2
LIVQTR_2_2.0	LIVQTR_2	v2
LIVQTR_2_3.0	LIVQTR_2	v2
LIVQTR_2_4.0	LIVQTR_2	v2
LIVQTR_3.0	LIVQTR	v1
LIVQTR_4.0	LIVQTR	v1
LIVQTR_5.0	LIVQTR	v1

	col_root	feat_group
LIVQTR_6.0	LIVQTR	v1
LIVQTR_7.0	LIVQTR	v1
LIVQTR_8.0	LIVQTR	v1
LIVQTR_9.0	LIVQTR	v1
MH_NOTGET_-1.0	MH_NOTGET	v1
MH_NOTGET_0.0	MH_NOTGET	dummy_drop
MH_NOTGET_1.0	MH_NOTGET	base
MH_SVCS_-1.0	MH_SVCS	v1
MH_SVCS_0.0	MH_SVCS	dummy_drop
MH_SVCS_1.0	MH_SVCS	base
MS_-1.0	MS	v1
MS_1.0	MS	dummy_drop
MS_2.0	MS	v1
MS_2_-1.0	MS_2	removed
MS_2_1.0	MS_2	dummy_drop
MS_2_2.0	MS_2	v2
MS_2_3.0	MS_2	v2
MS_3.0	MS	v1
MS_4.0	MS	v1
MS_5.0	MS	v1
NOTGET_-1.0	NOTGET	v1
NOTGET_0.0	NOTGET	dummy_drop
NOTGET_1.0	NOTGET	base
PLNDTRIPS_-1.0	PLNDTRIPS	v1
PLNDTRIPS_0.0	PLNDTRIPS	dummy_drop
PLNDTRIPS_1.0	PLNDTRIPS	base
PRESCRIPT_-1.0	PRESCRIPT	v1
PRESCRIPT_0.0	PRESCRIPT	dummy_drop
PRESCRIPT_1.0	PRESCRIPT	base
PROP_FOODSPEND_HOME	PROP_FOODSPEND_HOME	base
PROP_FOODSPEND_HOME_skipped_0.0	PROP_FOODSPEND_HOME	removed
PROP_FOODSPEND_HOME_skipped_1.0	PROP_FOODSPEND_HOME	v1
RHISPANIC_0.0	RHISPANIC	dummy_drop
RHISPANIC_1.0	RHISPANIC	base
RRACE_1.0	RRACE	dummy_drop

	col_root	feat_group
RRACE_2.0	RRACE	base
RRACE_3.0	RRACE	base
RRACE_4.0	RRACE	base
SCHOOL_KIDS_-1.0	SCHOOL_KIDS	removed
SCHOOL_KIDS_0.0	SCHOOL_KIDS	dummy_drop
SCHOOL_KIDS_1.0	SCHOOL_KIDS	v2
SNAP_YN_-1.0	SNAP_YN	v1
SNAP_YN_0.0	SNAP_YN	dummy_drop
SNAP_YN_1.0	SNAP_YN	base
SPNDSRC1_0.0	SPNDSRC1	dummy_drop
SPNDSRC1_1.0	SPNDSRC1	base
SPNDSRC2_0.0	SPNDSRC2	dummy_drop
SPNDSRC2_1.0	SPNDSRC2	base
SPNDSRC3_0.0	SPNDSRC3	dummy_drop
SPNDSRC3_1.0	SPNDSRC3	base
SPNDSRC4_0.0	SPNDSRC4	dummy_drop
SPNDSRC4_1.0	SPNDSRC4	base
SPNDSRC5_0.0	SPNDSRC5	dummy_drop
SPNDSRC5_1.0	SPNDSRC5	base
SPNDSRC6_0.0	SPNDSRC6	dummy_drop
SPNDSRC6_1.0	SPNDSRC6	base
SPNDSRC7_0.0	SPNDSRC7	dummy_drop
SPNDSRC7_1.0	SPNDSRC7	base
SSA_APPLY_-1.0	SSA_APPLY	v1
SSA_APPLY_0.0	SSA_APPLY	dummy_drop
SSA_APPLY_1.0	SSA_APPLY	base
SSA_RECV_-1.0	SSA_RECV	v1
SSA_RECV_0.0	SSA_RECV	dummy_drop
SSA_RECV_1.0	SSA_RECV	base
TENURE_-1.0	TENURE	v1
TENURE_1.0	TENURE	dummy_drop
TENURE_2.0	TENURE	base
TENURE_3.0	TENURE	base
TENURE_4.0	TENURE	base
THHLD_NUMADLT_1.0	THHLD_NUMADLT	dummy_drop

	col_root	feat_group
THHLD_NUMADLT_2.0	THHLD_NUMADLT	base
THHLD_NUMADLT_3.0	THHLD_NUMADLT	base
THHLD_NUMKID	THHLD_NUMKID	base
TNUM_PS	TNUM_PS	v1
TNUM_PS_2_-1.0	TNUM_PS_2	removed
TNUM_PS_2_0.0	TNUM_PS_2	dummy_drop
TNUM_PS_2_1.0	TNUM_PS_2	v2
TNUM_PS_skipped_0.0	TNUM_PS	removed
TNUM_PS_skipped_1.0	TNUM_PS	v1
TW_START_-1.0	TW_START	v1
TW_START_0.0	TW_START	dummy_drop
TW_START_1.0	TW_START	base
UI_APPLY_-1.0	UI_APPLY	v1
UI_APPLY_0.0	UI_APPLY	dummy_drop
UI_APPLY_1.0	UI_APPLY	base
WORRY_-1.0	WORRY	v1
WORRY_1.0	WORRY	dummy_drop
WORRY_2.0	WORRY	base
WORRY_3.0	WORRY	base
WORRY_4.0	WORRY	base
WRKLOSS_-1.0	WRKLOSS	v1
WRKLOSS_0.0	WRKLOSS	dummy_drop
WRKLOSS_1.0	WRKLOSS	base
inc_binary_0.0	inc_binary	dummy_drop
inc_binary_1.0	inc_binary	v2
incomplete	incomplete	v1
polit_0	polit	dummy_drop
polit_1	polit	base
skipped	skipped	v2

In [592…

```
# LIVQTR manually drop the most common scenario
manual_cols = ['LIVQTR', 'LIVQTR_2']
for col in manual_cols:
    # get most common label from original df
    most_common = str(df[col].value_counts().index[0])
    drop_ix = col + '_' + most_common
    # update df_feats to dummy that col
    df_feats.loc[df_feats.index == drop_ix, 'feat_group'] = 'dummy_drop'
```

```
df_feats.loc[df_feats['col_root'].isin(manual_cols)]
```

Out[592…

	col_root	feat_group
LIVQTR_-1.0	LIVQTR	v1
LIVQTR_1.0	LIVQTR	v1
LIVQTR_10.0	LIVQTR	v1
LIVQTR_2.0	LIVQTR	dummy_drop
LIVQTR_2_-1.0	LIVQTR_2	removed
LIVQTR_2_1.0	LIVQTR_2	v2
LIVQTR_2_2.0	LIVQTR_2	dummy_drop
LIVQTR_2_3.0	LIVQTR_2	v2
LIVQTR_2_4.0	LIVQTR_2	v2
LIVQTR_3.0	LIVQTR	v1
LIVQTR_4.0	LIVQTR	v1
LIVQTR_5.0	LIVQTR	v1
LIVQTR_6.0	LIVQTR	v1
LIVQTR_7.0	LIVQTR	v1
LIVQTR_8.0	LIVQTR	v1
LIVQTR_9.0	LIVQTR	v1

In [593…

```
# create lists of v1 and v2 cols from dataframe
v1_cols = df_feats.loc[df_feats['feat_group'].isin(['base', 'v1'])].index.to_lis
v2_cols = df_feats.loc[df_feats['feat_group'].isin(['base', 'v2'])].index.to_lis

v1_cols.sort()
v2_cols.sort()
```

In [594…

```
# Confirm columns that are in X_train but not in either v1 or v2 cols
# they should be only _skipped_0.0
# and _-1.0 versions of v2 features I engineered
# and also dummy OHE labels that I dropped
set(X_train_df.columns) - set(v1_cols + v2_cols)
```

Out[594…

```
{'ANXIOUS_1.0',
 'ANYWORK_0.0',
 'CHNGHOW10_0.0',
 'CHNGHOW11_0.0',
 'CHNGHOW12_0.0',
 'CHNGHOW1_0.0',
 'CHNGHOW2_0.0',
 'CHNGHOW3_0.0',
 'CHNGHOW4_0.0',
 'CHNGHOW5_0.0',
 'CHNGHOW6_0.0',
 'CHNGHOW7_0.0',
 'CHNGHOW8_0.0',
 'CHNGHOW9_0.0',
 'CURFOODSUF_skipped_0.0',
```

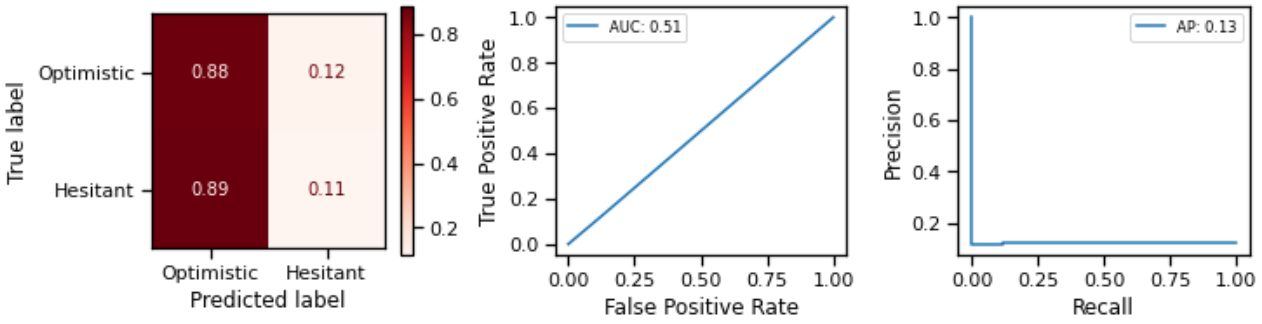
```
'DELAY_0.0',
'DOWN_1.0',
'EGENDER_0.0',
'EIP_1.0',
'EST_MSA_0.0',
'EXPCTLOSS_0.0',
'EXPNS_DIF_skipped_0.0',
'FEWRTRANS_1.0',
'FEWRTRIPS_0.0',
'FREEFOOD_0.0',
'HADCOVID_0.0',
'HLTHINS1_0.0',
'HLTHINS2_0.0',
'HLTHINS3_0.0',
'HLTHINS4_0.0',
'HLTHINS5_0.0',
'HLTHINS6_0.0',
'HLTHINS7_0.0',
'HLTHINS8_0.0',
'HLTHINS_-1.0',
'HLTHINS_0.0',
'INCOME_skipped_0.0',
'INTEREST_1.0',
'IN_METRO_AREA_0.0',
'LIVQTR_2.0',
'LIVQTR_2_-1.0',
'LIVQTR_2_2.0',
'MH_NOTGET_0.0',
'MH_SVCS_0.0',
'MS_1.0',
'MS_2_-1.0',
'MS_2_1.0',
'NOTGET_0.0',
'PLNDTRIPS_0.0',
'PRESCRIPT_0.0',
'PROP_FOODSPEND_HOME_skipped_0.0',
'RHISPANIC_0.0',
'RRACE_1.0',
'SCHOOL_KIDS_-1.0',
'SCHOOL_KIDS_0.0',
'SNAP_YN_0.0',
'SPNDSRC1_0.0',
'SPNDSRC2_0.0',
'SPNDSRC3_0.0',
'SPNDSRC4_0.0',
'SPNDSRC5_0.0',
'SPNDSRC6_0.0',
'SPNDSRC7_0.0',
'SSA_APPLY_0.0',
'SSA_RECV_0.0',
'TENURE_1.0',
'THHLD_NUMADLT_1.0',
'TNUM_PS_2_-1.0',
'TNUM_PS_2_0.0',
'TNUM_PS_skipped_0.0',
'TW_START_0.0',
'UI_APPLY_0.0',
'WORRY_1.0',
'WRKLOSS_0.0',
'inc_binary_0.0',
'polit_0'}
```

Create Dummy Model

In [595…
```
labels = ['Optimistic', 'Hesitant']
```

In [596…
```
# dummy classifier on non-SMOTEd data
dum = DummyClassifier(strategy='stratified')

dum.fit(X_train_df[v1_cols], y_train)

dstools.eval_clf_model(dum, X_test_df, y_test, X_train_df[v1_cols], y_train,
                       labels=labels)
```

```
****************************** Training Data ******************************
              precision    recall  f1-score   support

         0.0       0.88      0.88      0.88     51283
         1.0       0.12      0.12      0.12      7085

    accuracy                           0.79     58368
   macro avg       0.50      0.50      0.50     58368
weighted avg       0.79      0.79      0.79     58368


****************************** Test Data ******************************
              precision    recall  f1-score   support

         0.0       0.88      0.88      0.88     17037
         1.0       0.13      0.13      0.13      2420

    accuracy                           0.79     19457
   macro avg       0.51      0.51      0.51     19457
weighted avg       0.78      0.79      0.79     19457


****************************** Training Scores ******************************
           Training Macro F1 = 0.4982
        Training Macro Recall = 0.4982
   Training Balanced Accuracy = 0.4982

****************************** Test Scores ******************************
               Test Macro F1 = 0.5055
            Test Macro Recall = 0.5054
       Test Balanced Accuracy = 0.5054

****************************** Differences ******************************
         Train-Test Macro F1 Diff = 0.007299999999999973
      Train-Test Macro Recall Diff = 0.007199999999999984
Train-Test Balanced Accuracy Diff = 0.007199999999999984

****************************** Graphs for Test ******************************
```

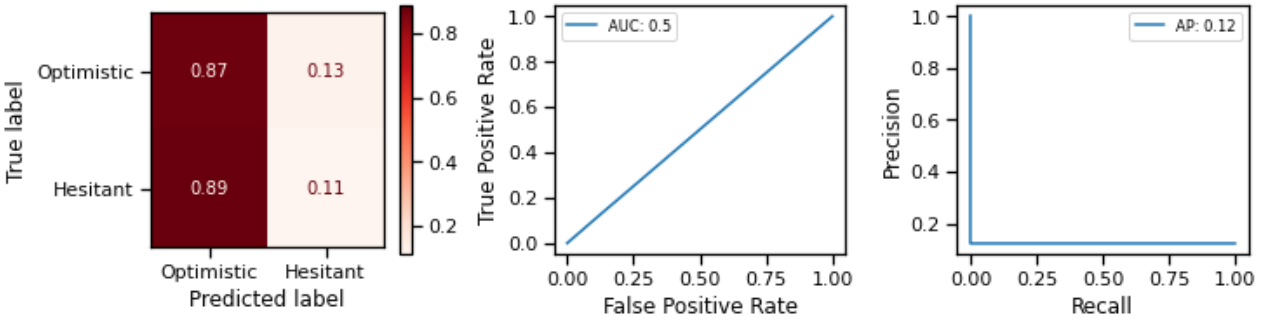


Performs appropriately poorly. Let's try on the SMOTEd data as well.

In [597…
```
dum_bal = DummyClassifier(strategy='stratified')
```

```
dum_bal.fit(X_train_sm[v1_cols], y_train_sm)

dstools.eval_clf_model(dum, X_test_df, y_test, X_train_sm[v1_cols], y_train_sm,
                       labels=labels)
```

```
****************************** Training Data ******************************
              precision    recall  f1-score   support

         0.0       0.50      0.88      0.64     51283
         1.0       0.50      0.12      0.20     51283

    accuracy                           0.50    102566
   macro avg       0.50      0.50      0.42    102566
weighted avg       0.50      0.50      0.42    102566


****************************** Test Data ******************************
              precision    recall  f1-score   support

         0.0       0.87      0.88      0.88     17037
         1.0       0.12      0.12      0.12      2420

    accuracy                           0.78     19457
   macro avg       0.50      0.50      0.50     19457
weighted avg       0.78      0.78      0.78     19457


****************************** Training Scores ******************************
            Training Macro F1 = 0.417
         Training Macro Recall = 0.5007
    Training Balanced Accuracy = 0.5007

****************************** Test Scores ******************************
                Test Macro F1 = 0.4968
             Test Macro Recall = 0.4968
        Test Balanced Accuracy = 0.4968

****************************** Differences ******************************
          Train-Test Macro F1 Diff = 0.07980000000000004
       Train-Test Macro Recall Diff = -0.0039000000000000146
Train-Test Balanced Accuracy Diff = -0.0039000000000000146

****************************** Graphs for Test ******************************
```



Also performs quite poorly. Great, now I can start modeling for real!

Initial Models

I'll start by creating initial vanilla models using both the SMOTEd data, and original unbalanced data with the `class_weight='balanced'` parameter.

I'll use the 'v1' columns, aka the columns most close to the original data. I think it makes sense to use these for my baseline, so I can perform feature selection and compare model performance with those features to the baselines.

This will serve two purposes: I want to get some baseline models to compare to later once I start tuning, and also see if I can suss out which class imbalance approach will be best for this data set.

Logistic Regression

Here I create two different logistic regression models without regularization, testing whether letting sklearn address class imbalance or using SMOTEd data is a better result.

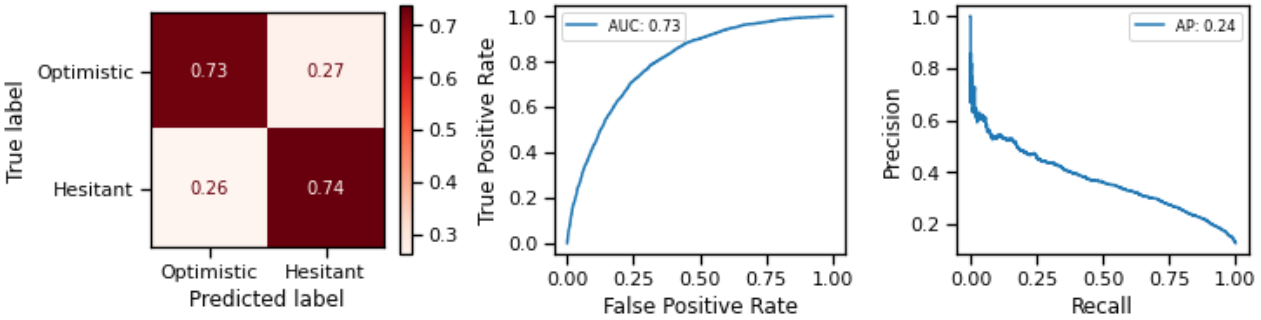
I initially used the default `lbfgs` solver and had to bump up the max iterations to 300 to get it to converge. When I tried to cross validate, the models failed to converge even with max iter of 500. I switched to `saga` solver, which worked better for this larger dataset.

In [598…

```
# unSMOTEd data
lr_ub = LogisticRegression(C=1e12, class_weight='balanced', max_iter=500,
                           solver='saga')

lr_ub.fit(X_train_df[v1_cols], y_train)

dstools.eval_clf_model(lr_ub, X_test_df[v1_cols], y_test, X_train_df[v1_cols],
                y_train, labels=labels)
```

```
****************************** Training Data ******************************
              precision    recall  f1-score   support

         0.0       0.95      0.72      0.82     51283
         1.0       0.27      0.75      0.39      7085

    accuracy                           0.72     58368
   macro avg       0.61      0.73      0.61     58368
weighted avg       0.87      0.72      0.77     58368


****************************** Test Data ******************************
              precision    recall  f1-score   support

         0.0       0.95      0.73      0.82     17037
         1.0       0.28      0.74      0.40      2420

    accuracy                           0.73     19457
   macro avg       0.61      0.73      0.61     19457
weighted avg       0.87      0.73      0.77     19457


****************************** Training Scores ******************************
            Training Macro F1 = 0.6073
         Training Macro Recall = 0.7323
   Training Balanced Accuracy = 0.7323

****************************** Test Scores ******************************
                Test Macro F1 = 0.6143
             Test Macro Recall = 0.733
        Test Balanced Accuracy = 0.733

****************************** Differences ******************************
```

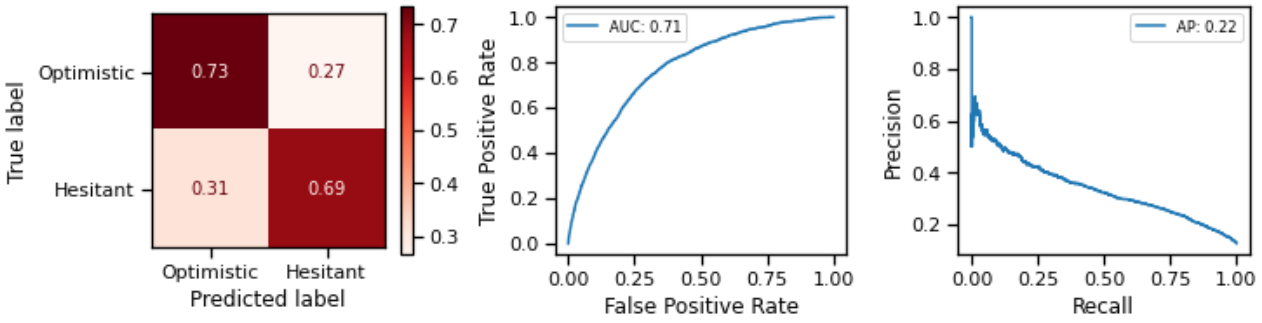
```
        Train-Test Macro F1 Diff = 0.007000000000000006
     Train-Test Macro Recall Diff = 0.0007000000000000339
Train-Test Balanced Accuracy Diff = 0.0007000000000000339

****************************** Graphs for Test ******************************
```



In [599…

```
# SMOTEd data
lr_sm = LogisticRegression(C=1e12, max_iter=500, solver='saga')

lr_sm.fit(X_train_sm[v1_cols], y_train_sm)

dstools.eval_clf_model(lr_sm, X_test_df[v1_cols], y_test, X_train_sm[v1_cols],
                       y_train_sm, labels=labels)
```

```
****************************** Training Data ******************************
              precision    recall  f1-score   support

         0.0       0.76      0.73      0.74     51283
         1.0       0.74      0.77      0.75     51283

    accuracy                           0.75    102566
   macro avg       0.75      0.75      0.75    102566
weighted avg       0.75      0.75      0.75    102566


****************************** Test Data ******************************
              precision    recall  f1-score   support

         0.0       0.94      0.73      0.83     17037
         1.0       0.27      0.69      0.39      2420

    accuracy                           0.73     19457
   macro avg       0.61      0.71      0.61     19457
weighted avg       0.86      0.73      0.77     19457


****************************** Training Scores ******************************
           Training Macro F1 = 0.7455
        Training Macro Recall = 0.7456
   Training Balanced Accuracy = 0.7456

****************************** Test Scores ******************************
               Test Macro F1 = 0.6064
            Test Macro Recall = 0.711
       Test Balanced Accuracy = 0.711

****************************** Differences ******************************
         Train-Test Macro F1 Diff = -0.1391
     Train-Test Macro Recall Diff = -0.034600000000000075
Train-Test Balanced Accuracy Diff = -0.034600000000000075

****************************** Graphs for Test ******************************
```



In [600…

```
# perform cross validation to get average scores
lr_ub_scores = cross_val_score(lr_ub, X_train_df[v1_cols], y_train,
                               scoring='f1_macro', cv=5, n_jobs=-1)

lr_sm_scores = cross_val_score(lr_sm, X_train_sm[v1_cols], y_train_sm,
                               scoring='f1_macro', cv=5, n_jobs=-1)
print('*** sklearn macro f1 ***')
print(lr_ub_scores)
print(lr_ub_scores.mean())
print()
print('*** SMOTE macro f1 ***')
print(lr_sm_scores)
print(lr_sm_scores.mean())
```

```
*** sklearn macro f1 ***
[0.59541909 0.60237545 0.61117774 0.60720976 0.60343198]
0.6039228025833753

*** SMOTE macro f1 ***
[0.70703256 0.74982745 0.75238449 0.74636736 0.75196689]
0.741515749884363
```

The vanilla model run on SMOTEd data did quite a bit better in terms of macro F1 using cross validation on training data.

However, with SMOTEd data there was a greater difference between F1 macro scores on train versus test. This looks like it's overfit. I'm concerned that by generating data using SMOTE, it has amplified noise in the training set that doesn't exist in the test set.

Random Forests

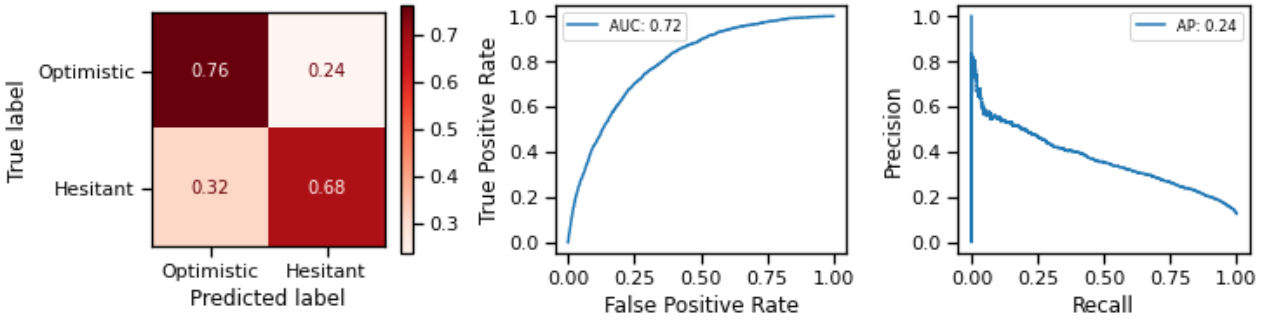
Starting with `max_depth =10` because the default parameter of `None` still resulted in a very overfit model.

In [601…

```
# unSMOTEd data
rf_ub = RandomForestClassifier(class_weight='balanced', n_jobs=-1,
                               max_depth=10)

rf_ub.fit(X_train_df[v1_cols], y_train)

dstools.eval_clf_model(rf_ub, X_test_df[v1_cols], y_test, X_train_df[v1_cols],
                       y_train, labels=labels)
```

```
****************************** Training Data ******************************
              precision    recall  f1-score   support

         0.0       0.96      0.77      0.86     51283
         1.0       0.32      0.78      0.46      7085
```

```
    accuracy                           0.77     58368
   macro avg       0.64      0.78      0.66     58368
weighted avg       0.88      0.77      0.81     58368


****************************** Test Data ******************************
              precision    recall  f1-score   support

         0.0       0.94      0.76      0.84     17037
         1.0       0.29      0.68      0.41      2420

    accuracy                           0.75     19457
   macro avg       0.62      0.72      0.63     19457
weighted avg       0.86      0.75      0.79     19457


****************************** Training Scores ******************************
            Training Macro F1 = 0.6559
         Training Macro Recall = 0.7777
    Training Balanced Accuracy = 0.7777

****************************** Test Scores ******************************
                Test Macro F1 = 0.6263
             Test Macro Recall = 0.7234
        Test Balanced Accuracy = 0.7234

****************************** Differences ******************************
          Train-Test Macro F1 Diff = -0.02960000000000007
       Train-Test Macro Recall Diff = -0.054299999999999904
Train-Test Balanced Accuracy Diff = -0.054299999999999904

****************************** Graphs for Test ******************************
```

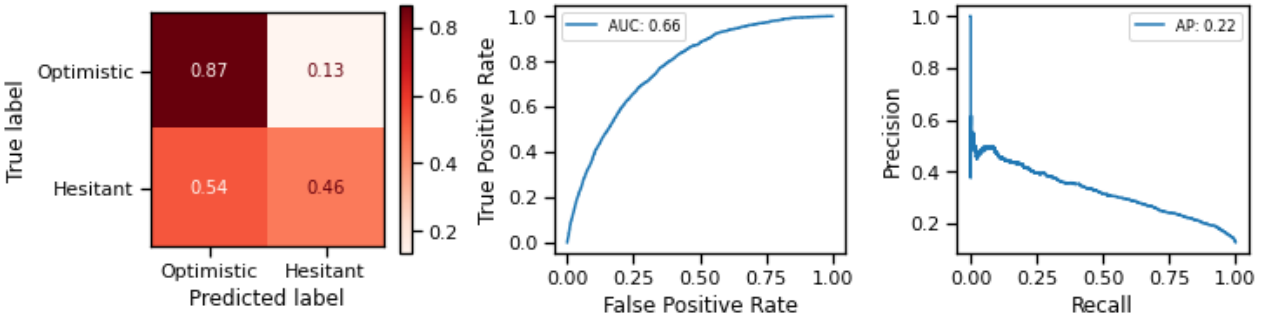


In [602…

```
# SMOTEd data
rf_sm = RandomForestClassifier(n_jobs=-1, max_depth=10)

rf_sm.fit(X_train_sm[v1_cols], y_train_sm)

dstools.eval_clf_model(rf_sm, X_test_df[v1_cols], y_test, X_train_sm[v1_cols],
                       y_train_sm, labels=labels)
```

```
****************************** Training Data ******************************
              precision    recall  f1-score   support

         0.0       0.91      0.87      0.89     51283
         1.0       0.88      0.91      0.89     51283

    accuracy                           0.89    102566
   macro avg       0.89      0.89      0.89    102566
weighted avg       0.89      0.89      0.89    102566
```

```
****************************** Test Data ******************************
              precision    recall  f1-score   support

         0.0       0.92      0.87      0.89     17037
         1.0       0.33      0.46      0.39      2420

    accuracy                           0.82     19457
   macro avg       0.63      0.66      0.64     19457
weighted avg       0.85      0.82      0.83     19457


****************************** Training Scores ******************************
           Training Macro F1 = 0.8927
        Training Macro Recall = 0.8927
   Training Balanced Accuracy = 0.8927

****************************** Test Scores ******************************
               Test Macro F1 = 0.6392
            Test Macro Recall = 0.6648
       Test Balanced Accuracy = 0.6648

****************************** Differences ******************************
         Train-Test Macro F1 Diff = -0.25350000000000006
     Train-Test Macro Recall Diff = -0.2279000000000001
Train-Test Balanced Accuracy Diff = -0.2279000000000001

****************************** Graphs for Test ******************************
```



In [603…

```
# cross validate
rf_ub_scores = cross_val_score(rf_ub, X_train_df[v1_cols], y_train,
                               scoring='f1_macro', cv=5, n_jobs=-1)

rf_sm_scores = cross_val_score(rf_sm, X_train_sm[v1_cols], y_train_sm,
                               scoring='f1_macro', cv=5, n_jobs=-1)
print('*** sklearn macro f1 ***')
print(rf_ub_scores)
print(rf_ub_scores.mean())
print()
print('*** SMOTE macro f1 ***')
print(rf_sm_scores)
print(rf_sm_scores.mean())
```

```
*** sklearn macro f1 ***
[0.61236863 0.61375684 0.62387509 0.62574672 0.62063496]
0.619276448100143

*** SMOTE macro f1 ***
[0.72933102 0.91573409 0.91754507 0.91270921 0.91358124]
0.8777801252112608
```

Decision Trees are prone to overfitting but a using a Random Forest should help combat that, especially with a max_depth parameter of only 10 considering I have many features.

I see the same pattern here between SMOTEd and weighted balancing where although SMOTEd data performs higher on training F1 macro, it looks way overfit on test.

I'm going proceed with using the original data and asking sklearn to balance it for me.

GridSearch for Best Params

Since I decided to go with original unbalanced data, I want to determine the best parameters for these models using v1 features before I proceed to reducing features.

In [126…
```
rerun_gridsearch = False
```

Logistic Regression

Although I chose F1 macro as my primary statistic, recall would also be a good choice, since I definitely want my model to find as many of the Hesitant class as possible.

I did a grid search optimizing for both statistics, and compared, to see how different the 'best' parameters would be.

In [605…
```
# Logistic regression, optimized for f1_macro
lr = LogisticRegression(max_iter=600, class_weight='balanced')

param_grid = {
    'C':[0.01, 1, 100, 1e6],
    'penalty': ['l1', 'l2', 'elasticnet'],
    'solver': ['liblinear', 'saga', 'lbfgs']
}

gs_lr = GridSearchCV(lr, param_grid, scoring='f1_macro', n_jobs=-1,  verbose=Tru

if rerun_gridsearch:
    gs_lr.fit(X_train_df[v1_cols], y_train)

    print(gs_lr.best_estimator_)
    print(gs_lr.best_score_)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   25.4s
[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed:  2.7min finished
LogisticRegression(C=1, class_weight='balanced', max_iter=600, solver='saga')
0.6040069636007855
```

In [606…
```
# Logistic regression, optimized for recall_macro
lr = LogisticRegression(max_iter=600, class_weight='balanced')

param_grid = {
    'C':[0.01, 1, 100, 1e6],
    'penalty': ['l1', 'l2', 'elasticnet'],
    'solver': ['liblinear', 'saga', 'lbfgs']
}

gs_lr = GridSearchCV(lr, param_grid, scoring='recall_macro', n_jobs=-1,
                     verbose=True)
```

```
if rerun_gridsearch:
    gs_lr.fit(X_train_df[v1_cols], y_train)

    print(gs_lr.best_estimator_)
    print(gs_lr.best_score_)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   20.5s
[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed:  2.5min finished
LogisticRegression(C=1, class_weight='balanced', max_iter=600, penalty='l1',
                   solver='liblinear')
0.7271066859697445
```

Best params for F1 macro:

- LogisticRegression(C=1, class_weight='balanced', max_iter=600, penalty='l2', solver='saga')
- 0.6046278796323324

Best params for recall macro:

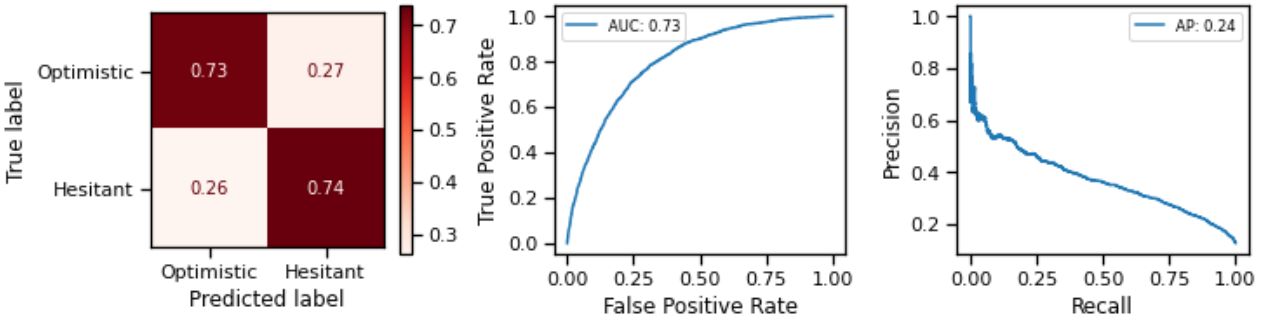
- LogisticRegression(C=1, class_weight='balanced', max_iter=600, penalty='l1', solver='liblinear')
- 0.7278955046732235

In [613…

```
# Check out results on test for recall optizmiation
lr = LogisticRegression(C=1, class_weight='balanced', max_iter=600,
                        penalty='l1', solver='liblinear')
lr.fit(X_train_df[v1_cols], y_train)

dstools.eval_clf_model(lr, X_test_df[v1_cols], y_test, X_train_df[v1_cols],
                y_train, labels=labels)
```

```
****************************** Training Data ******************************
              precision    recall  f1-score   support

         0.0       0.95      0.72      0.82     51283
         1.0       0.27      0.74      0.39      7085

    accuracy                           0.72     58368
   macro avg       0.61      0.73      0.61     58368
weighted avg       0.87      0.72      0.77     58368


****************************** Test Data ******************************
              precision    recall  f1-score   support

         0.0       0.95      0.73      0.82     17037
         1.0       0.28      0.74      0.40      2420

    accuracy                           0.73     19457
   macro avg       0.61      0.73      0.61     19457
weighted avg       0.87      0.73      0.77     19457


****************************** Training Scores ******************************
            Training Macro F1 = 0.6069
         Training Macro Recall = 0.7318
   Training Balanced Accuracy = 0.7318
```

```
****************************** Test Scores ******************************
               Test Macro F1 = 0.6142
            Test Macro Recall = 0.7328
       Test Balanced Accuracy = 0.7328

****************************** Differences ******************************
         Train-Test Macro F1 Diff = 0.007299999999999973
     Train-Test Macro Recall Diff = 0.0010000000000000009
Train-Test Balanced Accuracy Diff = 0.0010000000000000009

****************************** Graphs for Test ******************************
```

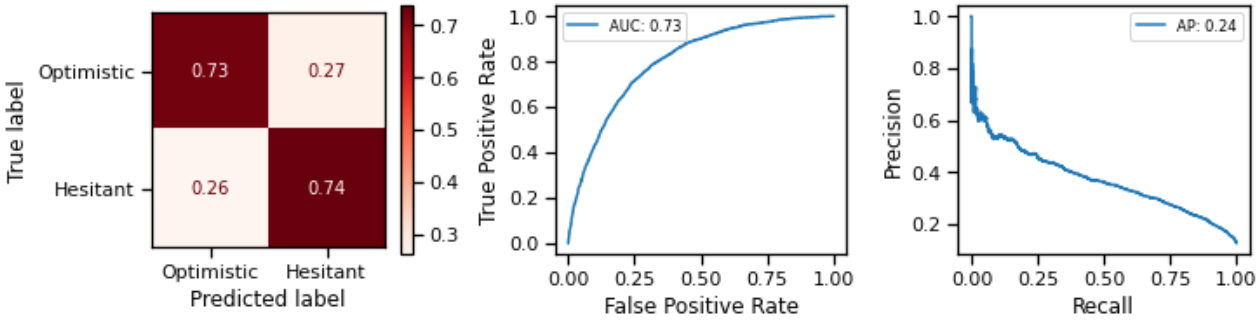


In [614…

```
# Check out results on test for F1 optizmiation
lr = LogisticRegression(C=1, class_weight='balanced', max_iter=600,
                        penalty='l2', solver='saga')
lr.fit(X_train_df[v1_cols], y_train)

dstools.eval_clf_model(lr, X_test_df[v1_cols], y_test, X_train_df[v1_cols],
               y_train, labels=labels)
```

```
****************************** Training Data ******************************
              precision    recall  f1-score   support

         0.0       0.95      0.72      0.82     51283
         1.0       0.27      0.75      0.39      7085

    accuracy                           0.72     58368
   macro avg       0.61      0.73      0.61     58368
weighted avg       0.87      0.72      0.77     58368


****************************** Test Data ******************************
              precision    recall  f1-score   support

         0.0       0.95      0.73      0.82     17037
         1.0       0.28      0.74      0.40      2420

    accuracy                           0.73     19457
   macro avg       0.61      0.73      0.61     19457
weighted avg       0.87      0.73      0.77     19457


****************************** Training Scores ******************************
           Training Macro F1 = 0.6073
        Training Macro Recall = 0.7324
   Training Balanced Accuracy = 0.7324

****************************** Test Scores ******************************
               Test Macro F1 = 0.6144
            Test Macro Recall = 0.7334
       Test Balanced Accuracy = 0.7334

****************************** Differences ******************************
```

```
        Train-Test Macro F1 Diff = 0.007099999999999995
     Train-Test Macro Recall Diff = 0.0010000000000000009
Train-Test Balanced Accuracy Diff = 0.0010000000000000009

****************************** Graphs for Test ******************************
```



These aren't much different, they're both pretty good, with very small differences between training and test.

I think F1 is a better balance, so I'm going to use those params.

Random Forest

In [609…

```
# Random Forest, optimized for f1_macro
rf = RandomForestClassifier(class_weight='balanced')

param_grid = {
    'criterion':['gini', 'entropy'],
    'max_depth': [5, 10, 30],
    'min_samples_split': [1, 5, 20],
    'min_impurity_decrease': [0, 0.01, 0.02],
    'max_features': [10, 20],
    'max_leaf_nodes': [6000, 2000, 500]
}

gs_rf = GridSearchCV(rf, param_grid, scoring='f1_macro', n_jobs=-1,
                     verbose=True)

if rerun_gridsearch:
    gs_rf.fit(X_train_df[v1_cols], y_train)

    print(gs_rf.best_estimator_)
    print(gs_rf.best_score_)
```

```
Fitting 5 folds for each of 324 candidates, totalling 1620 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  1.0min
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:  4.8min
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed: 11.3min
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed: 26.3min
[Parallel(n_jobs=-1)]: Done 1242 tasks      | elapsed: 40.8min
[Parallel(n_jobs=-1)]: Done 1620 out of 1620 | elapsed: 57.5min finished
RandomForestClassifier(class_weight='balanced', criterion='entropy',
                       max_depth=30, max_features=10, max_leaf_nodes=2000,
                       min_impurity_decrease=0, min_samples_split=20)
0.6557975680550348
```

Best RandomForest params optimizing for F1_macro:

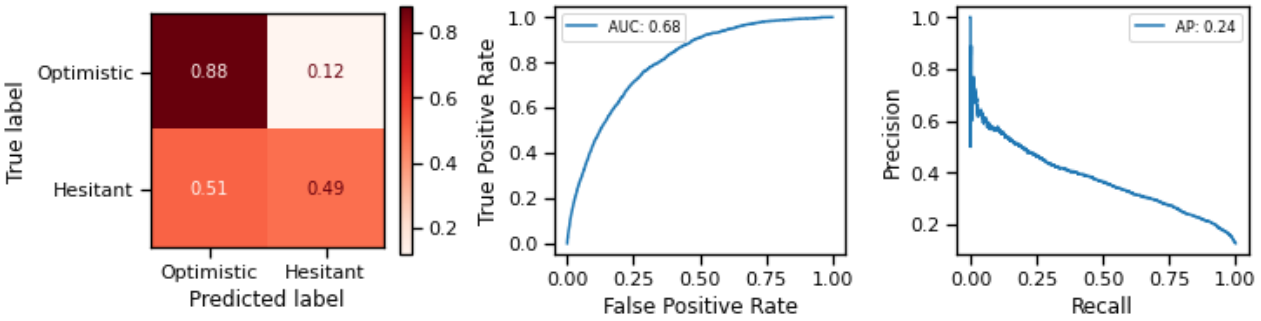
- RandomForestClassifier(class_weight='balanced', criterion='entropy', max_depth=30, max_features=10, max_leaf_nodes=2000, min_impurity_decrease=0, min_samples_split=20)
- 0.6557975680550348

This is pretty close to the F1 macro I got using only max_depth = 10.

In [617…

```
# Review model performance for a model with these optimal params
rf = RandomForestClassifier(class_weight='balanced', criterion='entropy',
                            max_depth=30, max_features=10,
                            max_leaf_nodes=2000, min_samples_split=20)


rf.fit(X_train_df[v1_cols], y_train)

dstools.eval_clf_model(rf, X_test_df[v1_cols], y_test, X_train_df[v1_cols],
               y_train, labels=labels)
```

```
****************************** Training Data ******************************
              precision    recall  f1-score   support

         0.0       0.98      0.91      0.94     51283
         1.0       0.57      0.88      0.69      7085

    accuracy                           0.91     58368
   macro avg       0.78      0.89      0.82     58368
weighted avg       0.93      0.91      0.91     58368


****************************** Test Data ******************************
              precision    recall  f1-score   support

         0.0       0.92      0.88      0.90     17037
         1.0       0.37      0.49      0.42      2420

    accuracy                           0.83     19457
   macro avg       0.65      0.68      0.66     19457
weighted avg       0.85      0.83      0.84     19457


****************************** Training Scores ******************************
            Training Macro F1 = 0.8192
         Training Macro Recall = 0.8936
    Training Balanced Accuracy = 0.8936

****************************** Test Scores ******************************
                Test Macro F1 = 0.6602
             Test Macro Recall = 0.6834
        Test Balanced Accuracy = 0.6834

****************************** Differences ******************************
          Train-Test Macro F1 Diff = -0.15900000000000003
       Train-Test Macro Recall Diff = -0.21019999999999994
Train-Test Balanced Accuracy Diff = -0.21019999999999994

****************************** Graphs for Test ******************************
```



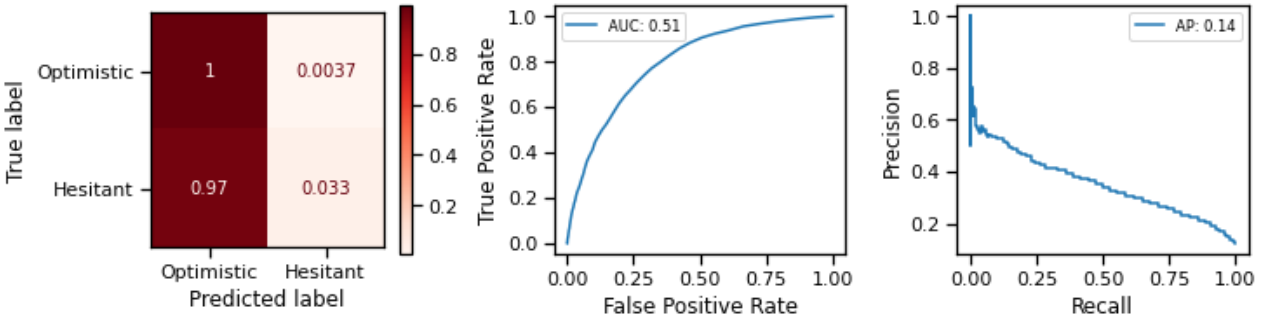
Not great on test. Let's take out some of the params that were at the high end of my grid search; maybe they were the best just because they were highest, and higher would actually be better.

In [618…

```
# Review model performance for a model with optimal params that weren't the
# max in the grid search.
rf = RandomForestClassifier(class_weight='balanced', criterion='entropy',
                            max_features=10)


rf.fit(X_train_df[v1_cols], y_train)

dstools.eval_clf_model(rf, X_test_df[v1_cols], y_test, X_train_df[v1_cols],
               y_train, labels=labels)
```

```
****************************** Training Data ******************************
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     51283
         1.0       1.00      1.00      1.00      7085

    accuracy                           1.00     58368
   macro avg       1.00      1.00      1.00     58368
weighted avg       1.00      1.00      1.00     58368


****************************** Test Data ******************************
              precision    recall  f1-score   support

         0.0       0.88      1.00      0.93     17037
         1.0       0.56      0.03      0.06      2420

    accuracy                           0.88     19457
   macro avg       0.72      0.51      0.50     19457
weighted avg       0.84      0.88      0.83     19457


****************************** Training Scores ******************************
            Training Macro F1 = 1.0
         Training Macro Recall = 0.9999
    Training Balanced Accuracy = 0.9999

****************************** Test Scores ******************************
                Test Macro F1 = 0.4982
             Test Macro Recall = 0.5147
        Test Balanced Accuracy = 0.5147

****************************** Differences ******************************
          Train-Test Macro F1 Diff = -0.5018
       Train-Test Macro Recall Diff = -0.48519999999999996
Train-Test Balanced Accuracy Diff = -0.48519999999999996
```

```
****************************** Graphs for Test ******************************
```



Definitely not doing well on test still.

Maybe I'm overthinking it trying too many parameters in my grid search. I'll simplify it to just `criterion` and `max_depth`.

In [612…

```
# How about trying another grid search with just max_depth and criterion?
rf = RandomForestClassifier(class_weight='balanced')

param_grid = {
    'criterion':['gini', 'entropy'],
    'max_depth': [5, 10, 30]
}

gs_rf = GridSearchCV(rf, param_grid, scoring='f1_macro', n_jobs=-1,
                     verbose=True)

if rerun_gridsearch:
    gs_rf.fit(X_train_df[v1_cols], y_train)

    print(gs_rf.best_estimator_)
    print(gs_rf.best_score_)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  2.9min finished
RandomForestClassifier(class_weight='balanced', max_depth=10)
0.6186557191120258
```

This time I got:

- RandomForestClassifier(class_weight='balanced', max_depth=10)
- 0.624889713409605

Which is what I had tried before and did pretty well. I don't think trying to fine tune more parameters using grid search is helping.

Feature Selection

I have some decent baseline models using the v1_cols that are closest to the originals.

Now I'll go through a feature selection process considering ALL the columns, to narrow down the feature space. This process should also help identify if any of the features I engineered might be useful to sub in for the originals.

In [619…
```
v1_v2_cols = list(set(v1_cols + v2_cols))
```

Variance Threshold

In [620…
```
# I'm pretty sure from my EDA that I don't have any features with 0 variance
# so I'm going to start with a threshold of .95

# since most of my variables are OHE, I'll use the formula p(1-p) to generate
# the appropriate variace threshold
# More here: (https://scikit-learn.org/stable/modules/feature_selection.html)
threshold = 0.99*(1-0.99)

sel = VarianceThreshold(threshold)
sel.fit(X_train_df[v1_v2_cols])

# Review columns to be removed
list(X_train_df[v1_v2_cols].loc[:, sel.get_support()==False].head().columns)
```

Out[620…
```
['MH_NOTGET_-1.0',
 'ANYWORK_-1.0',
 'LIVQTR_-1.0',
 'SSA_RECV_-1.0',
 'SNAP_YN_-1.0',
 'PLNDTRIPS_-1.0',
 'ANXIOUS_-1.0',
 'SSA_APPLY_-1.0',
 'EXPCTLOSS_-1.0',
 'FEWRTRANS_-1.0',
 'FEWRTRIPS_-1.0',
 'HLTHINS7_1.0',
 'FREEFOOD_-1.0',
 'CURFOODSUF_skipped_1.0',
 'EXPNS_DIF_skipped_1.0',
 'MS_-1.0',
 'INTEREST_-1.0',
 'LIVQTR_10.0',
 'EIP_-1.0',
 'WRKLOSS_-1.0',
 'DOWN_-1.0',
 'NOTGET_-1.0',
 'LIVQTR_2_4.0',
 'UI_APPLY_-1.0',
 'DELAY_-1.0',
 'HADCOVID_-1.0',
 'PRESCRIPT_-1.0',
 'skipped',
 'MH_SVCS_-1.0',
 'TENURE_-1.0',
 'WORRY_-1.0']
```

Almost all of these are skipped indicator columns for specific questions, which makes sense because those are sparsely populated.

I'll come back to this if the other processes don't filter these features out using their methods.

Univariate Feature Selection

There are 3 different algorithms to univariate feature selection methods that are suitable for classification problems. I decided to try each one out and give them an equal vote in terms of which features to keep.

For each algorithm, I asked for `k=50` features.

In [621…
```
# using mutual info
selk = SelectKBest(mutual_info_classif, k=50)
selk.fit(X_train_df[v1_v2_cols], y_train)

# Review features that it would keep
keep50_mutual = list(X_train_df[v1_v2_cols].loc[:, selk.get_support()]\
                     .head().columns)
keep50_mutual
```

Out[621…
```
['HLTHINS4_-1.0',
 'IN_METRO_AREA_1.0',
 'LIVQTR_2_3.0',
 'RRACE_3.0',
 'EEDUC',
 'HLTHINS3_1.0',
 'CHNGHOW1_1.0',
 'ENROLL2_-1.0',
 'EIP_4.0',
 'THHLD_NUMADLT_3.0',
 'SPNDSRC1_1.0',
 'THHLD_NUMKID',
 'UI_APPLY_1.0',
 'HLTHINS1_1.0',
 'WRKLOSS_1.0',
 'CHNGHOW4_1.0',
 'polit_1',
 'FEWRTRIPS_1.0',
 'ANYWORK_1.0',
 'DOWN_3.0',
 'HLTHINS4_1.0',
 'TW_START_1.0',
 'CHNGHOW6_1.0',
 'PROP_FOODSPEND_HOME',
 'SPNDSRC2_1.0',
 'CURFOODSUF',
 'CHNGHOW8_1.0',
 'RRACE_4.0',
 'WORRY_4.0',
 'AGE',
 'EST_MSA_42660.0',
 'LIVQTR_1.0',
 'SPNDSRC4_1.0',
 'NOTGET_-1.0',
 'LIVQTR_2_4.0',
 'SNAP_YN_1.0',
 'CHNGHOW12_1.0',
 'PLNDTRIPS_1.0',
 'ANXIOUS_2.0',
 'THHLD_NUMADLT_2.0',
 'CHNGHOW2_1.0',
 'INCOME',
 'FEWRTRANS_3.0',
 'SSA_RECV_1.0',
 'HLTHINS_1.0',
 'LIVQTR_4.0',
 'EXPNS_DIF',
 'INTEREST_2.0',
 'ENROLL3_-1.0',
 'ENROLL1_1.0']
```

In [622…
```
# Using chi2
```

```
selk2 = SelectKBest(chi2, k=50)
selk2.fit(X_train_df[v1_v2_cols], y_train)

# Review features that it would keep
keep50_chi2 = list(X_train_df[v1_v2_cols].loc[:, selk2.get_support()]\
                   .head().columns)
keep50_chi2
```

Out[622…
```
['EIP_3.0',
 'HLTHINS8_-1.0',
 'HLTHINS4_-1.0',
 'IN_METRO_AREA_1.0',
 'RRACE_3.0',
 'EEDUC',
 'HLTHINS3_1.0',
 'CHNGHOW1_1.0',
 'EXPCTLOSS_1.0',
 'ENROLL2_-1.0',
 'HLTHINS7_-1.0',
 'DOWN_4.0',
 'SPNDSRC1_1.0',
 'THHLD_NUMKID',
 'UI_APPLY_1.0',
 'TENURE_3.0',
 'EST_MSA_41860.0',
 'WRKLOSS_1.0',
 'CHNGHOW4_1.0',
 'polit_1',
 'LIVQTR_2_1.0',
 'FEWRTRIPS_1.0',
 'HLTHINS4_1.0',
 'TW_START_1.0',
 'LIVQTR_10.0',
 'CHNGHOW6_1.0',
 'CHNGHOW9_1.0',
 'CURFOODSUF',
 'RRACE_4.0',
 'AGE',
 'LIVQTR_1.0',
 'SCHOOL_KIDS_1.0',
 'SPNDSRC4_1.0',
 'LIVQTR_2_4.0',
 'SNAP_YN_1.0',
 'CHNGHOW12_1.0',
 'PLNDTRIPS_1.0',
 'HLTHINS5_-1.0',
 'ANXIOUS_2.0',
 'ENROLL2_1.0',
 'CHNGHOW2_1.0',
 'INCOME',
 'HLTHINS6_-1.0',
 'SSA_RECV_1.0',
 'HADCOVID_1.0',
 'MS_2_3.0',
 'MS_5.0',
 'EXPNS_DIF',
 'ENROLL3_-1.0',
 'ENROLL1_1.0']
```

In [623…
```
# Using f
selk3 = SelectKBest(f_classif, k=50)
selk3.fit(X_train_df[v1_v2_cols], y_train)

# Review features that it would keep
```

```
keep50_f = list(X_train_df[v1_v2_cols].loc[:, selk3.get_support()]\
                .head().columns)
keep50_f
```

Out[623…
```
['EIP_3.0',
 'WORRY_2.0',
 'HLTHINS4_-1.0',
 'IN_METRO_AREA_1.0',
 'RRACE_3.0',
 'EEDUC',
 'HLTHINS3_1.0',
 'CHNGHOW1_1.0',
 'EXPCTLOSS_1.0',
 'ENROLL2_-1.0',
 'EIP_4.0',
 'HLTHINS7_-1.0',
 'SPNDSRC1_1.0',
 'THHLD_NUMKID',
 'UI_APPLY_1.0',
 'TENURE_3.0',
 'HLTHINS1_1.0',
 'WRKLOSS_1.0',
 'CHNGHOW4_1.0',
 'polit_1',
 'LIVQTR_2_1.0',
 'FEWRTRIPS_1.0',
 'HLTHINS4_1.0',
 'TW_START_1.0',
 'CHNGHOW6_1.0',
 'CHNGHOW9_1.0',
 'CURFOODSUF',
 'RRACE_4.0',
 'AGE',
 'LIVQTR_1.0',
 'SCHOOL_KIDS_1.0',
 'SPNDSRC4_1.0',
 'SNAP_YN_1.0',
 'CHNGHOW12_1.0',
 'PLNDTRIPS_1.0',
 'HLTHINS5_-1.0',
 'ANXIOUS_2.0',
 'ENROLL2_1.0',
 'CHNGHOW2_1.0',
 'INCOME',
 'FEWRTRANS_3.0',
 'HLTHINS6_-1.0',
 'SSA_RECV_1.0',
 'HLTHINS_1.0',
 'HADCOVID_1.0',
 'MS_2_3.0',
 'MS_5.0',
 'EXPNS_DIF',
 'ENROLL3_-1.0',
 'ENROLL1_1.0']
```

Select From Model - Logit

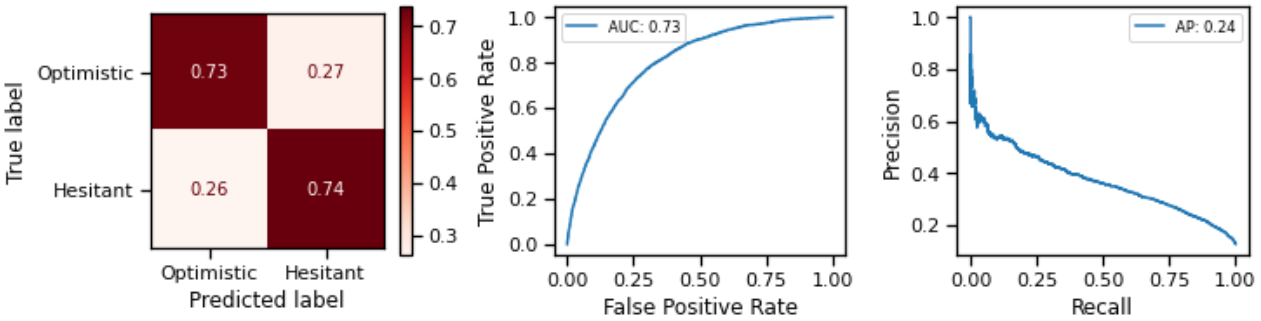
When I ran logistic regression, it found that L2 was the best performing regularization technique with `saga` solver, but now I'll use L1 Lasso with a to see how well it can reduce features for me.

I also tried this with a LinearSVC model, but even at 4000 max iterations, it failed to converge.

In [624…

```
# fitting on all columns, since will be using for feature selection
# Using best CV params optimizing for recall since it performed as well
# as f1 macro's best but using L1
lr = LogisticRegression(C=1, class_weight='balanced', max_iter=600,
                        penalty='l1', solver='liblinear')
lr.fit(X_train_df[v1_v2_cols], y_train)

dstools.eval_clf_model(lr, X_test_df[v1_v2_cols], y_test,
                       X_train_df[v1_v2_cols], y_train, labels=labels)
```

```
****************************** Training Data ******************************
              precision    recall  f1-score   support

         0.0       0.95      0.72      0.82     51283
         1.0       0.27      0.74      0.39      7085

    accuracy                           0.72     58368
   macro avg       0.61      0.73      0.61     58368
weighted avg       0.87      0.72      0.77     58368


****************************** Test Data ******************************
              precision    recall  f1-score   support

         0.0       0.95      0.73      0.82     17037
         1.0       0.28      0.74      0.40      2420

    accuracy                           0.73     19457
   macro avg       0.61      0.73      0.61     19457
weighted avg       0.87      0.73      0.77     19457


****************************** Training Scores ******************************
            Training Macro F1 = 0.6074
         Training Macro Recall = 0.7319
    Training Balanced Accuracy = 0.7319

****************************** Test Scores ******************************
                Test Macro F1 = 0.6147
             Test Macro Recall = 0.7333
        Test Balanced Accuracy = 0.7333

****************************** Differences ******************************
          Train-Test Macro F1 Diff = 0.007299999999999973
       Train-Test Macro Recall Diff = 0.0013999999999999568
Train-Test Balanced Accuracy Diff = 0.0013999999999999568

****************************** Graphs for Test ******************************
```



In [625…

```
model_sel = SelectFromModel(lr, prefit=True)
model_sel.get_support()
```

Out[625…

```
array([ True,  True,  True,  True,  True, False,  True,  True,  True,
```

```
       True,  True,  True,  True,  True,  True,  True,  True,  True,
       True,  True,  True,  True,  True,  True,  True,  True,  True,
       True,  True,  True,  True,  True,  True,  True,  True, False,
       True,  True,  True,  True,  True,  True,  True,  True,  True,
       True,  True,  True,  True,  True,  True,  True,  True,  True,
       True,  True,  True,  True,  True,  True,  True,  True,  True,
       True,  True,  True,  True,  True,  True,  True,  True,  True,
       True,  True,  True,  True,  True,  True,  True,  True,  True,
       True,  True,  True,  True, False,  True,  True,  True,  True,
       True,  True,  True,  True,  True,  True,  True,  True,  True,
       True,  True,  True,  True,  True,  True,  True,  True,  True,
       True,  True,  True,  True,  True,  True,  True,  True,  True,
       True,  True,  True,  True,  True,  True,  True, False,  True,
       True, False,  True,  True, False,  True,  True,  True,  True,
       True,  True,  True,  True,  True,  True,  True,  True,  True,
      False,  True,  True,  True,  True,  True,  True,  True,  True,
       True,  True,  True])
```

In [626…

```
# what would it get rid of?
keep_lr = list(X_train_df[v1_v2_cols].loc[:, model_sel.get_support()==False]\
               .head().columns)
keep_lr
```

Out[626…

```
['MS_4.0',
 'FEWRTRANS_-1.0',
 'WORRY_4.0',
 'MS_2.0',
 'skipped',
 'EST_MSA_35620.0',
 'LIVQTR_4.0']
```

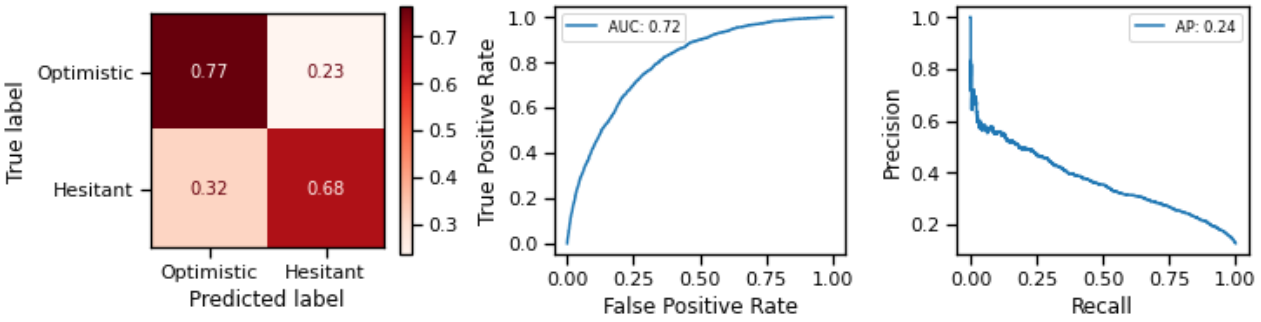
Select from Model - Forest

In [627…

```
# Using best minimal params from grid search
rf = RandomForestClassifier(max_depth=10, class_weight='balanced')

rf.fit(X_train_df[v1_v2_cols], y_train)

dstools.eval_clf_model(rf, X_test_df[v1_v2_cols], y_test,
               X_train_df[v1_v2_cols], y_train, labels=labels)
```

```
****************************** Training Data ******************************
              precision    recall  f1-score   support

         0.0       0.96      0.77      0.86     51283
         1.0       0.32      0.78      0.46      7085

    accuracy                           0.78     58368
   macro avg       0.64      0.78      0.66     58368
weighted avg       0.89      0.78      0.81     58368


****************************** Test Data ******************************
              precision    recall  f1-score   support

         0.0       0.94      0.77      0.85     17037
         1.0       0.29      0.68      0.41      2420

    accuracy                           0.75     19457
   macro avg       0.62      0.72      0.63     19457
weighted avg       0.86      0.75      0.79     19457
```

```
****************************** Training Scores ******************************
           Training Macro F1 = 0.658
        Training Macro Recall = 0.7782
   Training Balanced Accuracy = 0.7782

****************************** Test Scores ******************************
               Test Macro F1 = 0.6264
            Test Macro Recall = 0.722
       Test Balanced Accuracy = 0.722

****************************** Differences ******************************
         Train-Test Macro F1 Diff = -0.03160000000000007
     Train-Test Macro Recall Diff = -0.05620000000000003
Train-Test Balanced Accuracy Diff = -0.05620000000000003

****************************** Graphs for Test ******************************
```



In [628…
```
model_sel_f = SelectFromModel(rf, prefit=True)
model_sel_f.get_support()
```

Out[628…
```
array([False, False, False, False, False, False, False, False, False,
       False, False, False, False,  True, False, False,  True, False,
       False, False,  True, False,  True, False, False,  True, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False,  True, False,
       False, False, False,  True, False,  True, False, False, False,
       False, False, False,  True, False, False, False,  True, False,
        True, False,  True, False, False, False,  True, False,  True,
       False, False, False, False,  True, False, False,  True, False,
       False, False, False,  True, False, False, False, False, False,
       False, False, False, False, False,  True, False, False, False,
       False, False, False, False, False, False,  True, False, False,
       False, False, False, False, False,  True,  True,  True, False,
       False,  True, False, False, False, False, False, False, False,
       False, False, False,  True,  True, False, False, False, False,
        True,  True, False, False, False, False, False,  True, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False,  True, False, False, False, False, False,
       False,  True,  True])
```

In [629…
```
# what would it keep?
keep_tree = list(X_train_df[v1_v2_cols].loc[:, model_sel_f.get_support()]\
                 .head().columns)
keep_tree
```

Out[629…
```
['IN_METRO_AREA_1.0',
 'RRACE_3.0',
 'EEDUC',
 'HLTHINS3_1.0',
 'CHNGHOW1_1.0',
 'SPNDSRC1_1.0',
 'THHLD_NUMKID',
 'UI_APPLY_1.0',
```

```
'WRKLOSS_1.0',
'CHNGHOW4_1.0',
'polit_1',
'FEWRTRIPS_1.0',
'HLTHINS4_1.0',
'TW_START_1.0',
'CHNGHOW6_1.0',
'PROP_FOODSPEND_HOME',
'CURFOODSUF',
'AGE',
'SCHOOL_KIDS_1.0',
'SNAP_YN_1.0',
'CHNGHOW12_1.0',
'PLNDTRIPS_1.0',
'ANXIOUS_2.0',
'CHNGHOW2_1.0',
'INCOME',
'FEWRTRANS_3.0',
'skipped',
'SSA_RECV_1.0',
'EXPNS_DIF',
'ENROLL3_-1.0',
'ENROLL1_1.0']
```

Combine Votes for Features

This looks pretty reasonable.

Let's compare features across the different feature selection methods and see which are most commonly kept.

In [630…

```
# create dataframe with selected features from each process
feat_dict = {
    'univ_mutual' : list(selk.get_support()),
    'univ_chi2' : list(selk2.get_support()),
    'univ_f' : list(selk3.get_support()),
    'model_lr' : list(model_sel.get_support()),
    'model_rf' : list(model_sel_f.get_support())
    }

feat_votes_ib = pd.DataFrame.from_dict(feat_dict, orient='index',
                                       columns=v1_v2_cols)

# convert True to 1 and False to 0
# really useful:
#https://stackoverflow.com/questions/17383094/how-can-i-map-true-false-to-1-0-in
feat_votes_ib = feat_votes_ib*1


feat_votes_ib
```

Out[630…

	EIP_3.0	FREEFOOD_1.0	SPNDSRC7_1.0	TENURE_2.0	INCOME_skipped_1.0	MS_4.0
univ_mutual	0	0	0	0	0	0
univ_chi2	1	0	0	0	0	0
univ_f	1	0	0	0	0	0
model_lr	1	1	1	1	1	0
model_rf	0	0	0	0	0	0

5 rows × 165 columns

In [631…
```
# sum votes for each column
votes_ib = feat_votes_ib.sum(axis=0)
sorted_votes_ib = votes_ib.sort_values(ascending=False)
print(list(sorted_votes_ib))
```

```
[5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
5, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
0, 0, 0, 0]
```

It looks like there are a decent number of features that at least 2 of these feature selection methods voted for. I'm inclined to go with those, but let's take a look at what will be dropped if we exclude features that only 1 or no methods voted for.

In [632…
```
# What are the features that only 1 process voted for?
# Are they all different processes?
feat_votes_ib[sorted_votes_ib.loc[sorted_votes_ib == 1].index].sum(axis=1)
```

Out[632…
```
univ_mutual     2
univ_chi2       0
univ_f          0
model_lr       92
model_rf        1
dtype: int64
```

So the logistic regression model was the main one that voted for features which no other process voted for.

This makes sense, since that model used L1 regularization to try to reduce feature coefficients to 0, and it may not have been able to do that for as many.

In [633…
```
# These columns would be kept
to_keep_ib = list(sorted_votes_ib.loc[sorted_votes_ib >= 2].index)
to_keep_ib.sort()

print(len(to_keep_ib))
to_keep_ib
```

```
66
```

Out[633…
```
['AGE',
 'ANXIOUS_2.0',
 'ANYWORK_1.0',
 'CHNGHOW12_1.0',
 'CHNGHOW1_1.0',
 'CHNGHOW2_1.0',
 'CHNGHOW4_1.0',
 'CHNGHOW6_1.0',
 'CHNGHOW8_1.0',
 'CHNGHOW9_1.0',
 'CURFOODSUF',
 'DOWN_3.0',
 'DOWN_4.0',
 'EEDUC',
 'EIP_3.0',
```

```
'EIP_4.0',
'ENROLL1_1.0',
'ENROLL2_-1.0',
'ENROLL2_1.0',
'ENROLL3_-1.0',
'EST_MSA_41860.0',
'EST_MSA_42660.0',
'EXPCTLOSS_1.0',
'EXPNS_DIF',
'FEWRTRANS_3.0',
'FEWRTRIPS_1.0',
'HADCOVID_1.0',
'HLTHINS1_1.0',
'HLTHINS3_1.0',
'HLTHINS4_-1.0',
'HLTHINS4_1.0',
'HLTHINS5_-1.0',
'HLTHINS6_-1.0',
'HLTHINS7_-1.0',
'HLTHINS8_-1.0',
'HLTHINS_1.0',
'INCOME',
'INTEREST_2.0',
'IN_METRO_AREA_1.0',
'LIVQTR_1.0',
'LIVQTR_10.0',
'LIVQTR_2_1.0',
'LIVQTR_2_3.0',
'LIVQTR_2_4.0',
'MS_2_3.0',
'MS_5.0',
'NOTGET_-1.0',
'PLNDTRIPS_1.0',
'PROP_FOODSPEND_HOME',
'RRACE_3.0',
'RRACE_4.0',
'SCHOOL_KIDS_1.0',
'SNAP_YN_1.0',
'SPNDSRC1_1.0',
'SPNDSRC2_1.0',
'SPNDSRC4_1.0',
'SSA_RECV_1.0',
'TENURE_3.0',
'THHLD_NUMADLT_2.0',
'THHLD_NUMADLT_3.0',
'THHLD_NUMKID',
'TW_START_1.0',
'UI_APPLY_1.0',
'WORRY_2.0',
'WRKLOSS_1.0',
'polit_1']
```

In [634…

```
# Mark direct selections in the feat_df

# update direct selections
df_feats.loc[df_feats.index.isin(to_keep_ib), 'selected'] = 'direct'

# add in family members
#selected_roots = df_feats.loc[df_feats['selected']=='direct', 'col_root']
#df_feats.loc[(df_feats['col_root'].isin(selected_roots)) &
#             (df_feats['selected'].isna())
#             , 'selected'] = 'fam'

# This is the df filtered for only selected features
```

```
with pd.option_context('display.max_rows', 200):
    display(df_feats.loc[df_feats['selected'].isin(['direct'])])
```

	col_root	feat_group	selected
AGE	AGE	base	direct
ANXIOUS_2.0	ANXIOUS	base	direct
ANYWORK_1.0	ANYWORK	base	direct
CHNGHOW12_1.0	CHNGHOW12	base	direct
CHNGHOW1_1.0	CHNGHOW1	base	direct
CHNGHOW2_1.0	CHNGHOW2	base	direct
CHNGHOW4_1.0	CHNGHOW4	base	direct
CHNGHOW6_1.0	CHNGHOW6	base	direct
CHNGHOW8_1.0	CHNGHOW8	base	direct
CHNGHOW9_1.0	CHNGHOW9	base	direct
CURFOODSUF	CURFOODSUF	base	direct
DOWN_3.0	DOWN	base	direct
DOWN_4.0	DOWN	base	direct
EEDUC	EEDUC	base	direct
EIP_3.0	EIP	base	direct
EIP_4.0	EIP	base	direct
ENROLL1_1.0	ENROLL1	v1	direct
ENROLL2_-1.0	ENROLL2	v1	direct
ENROLL2_1.0	ENROLL2	v1	direct
ENROLL3_-1.0	ENROLL3	v1	direct
EST_MSA_41860.0	EST_MSA	v1	direct
EST_MSA_42660.0	EST_MSA	v1	direct
EXPCTLOSS_1.0	EXPCTLOSS	base	direct
EXPNS_DIF	EXPNS_DIF	base	direct
FEWRTRANS_3.0	FEWRTRANS	base	direct
FEWRTRIPS_1.0	FEWRTRIPS	base	direct
HADCOVID_1.0	HADCOVID	base	direct
HLTHINS1_1.0	HLTHINS1	v1	direct
HLTHINS3_1.0	HLTHINS3	v1	direct
HLTHINS4_-1.0	HLTHINS4	v1	direct
HLTHINS4_1.0	HLTHINS4	v1	direct
HLTHINS5_-1.0	HLTHINS5	v1	direct
HLTHINS6_-1.0	HLTHINS6	v1	direct

	col_root	feat_group	selected
HLTHINS7_-1.0	HLTHINS7	v1	direct
HLTHINS8_-1.0	HLTHINS8	v1	direct
HLTHINS_1.0	HLTHINS	v2	direct
INCOME	INCOME	base	direct
INTEREST_2.0	INTEREST	base	direct
IN_METRO_AREA_1.0	IN_METRO_AREA	v2	direct
LIVQTR_1.0	LIVQTR	v1	direct
LIVQTR_10.0	LIVQTR	v1	direct
LIVQTR_2_1.0	LIVQTR_2	v2	direct
LIVQTR_2_3.0	LIVQTR_2	v2	direct
LIVQTR_2_4.0	LIVQTR_2	v2	direct
MS_2_3.0	MS_2	v2	direct
MS_5.0	MS	v1	direct
NOTGET_-1.0	NOTGET	v1	direct
PLNDTRIPS_1.0	PLNDTRIPS	base	direct
PROP_FOODSPEND_HOME	PROP_FOODSPEND_HOME	base	direct
RRACE_3.0	RRACE	base	direct
RRACE_4.0	RRACE	base	direct
SCHOOL_KIDS_1.0	SCHOOL_KIDS	v2	direct
SNAP_YN_1.0	SNAP_YN	base	direct
SPNDSRC1_1.0	SPNDSRC1	base	direct
SPNDSRC2_1.0	SPNDSRC2	base	direct
SPNDSRC4_1.0	SPNDSRC4	base	direct
SSA_RECV_1.0	SSA_RECV	base	direct
TENURE_3.0	TENURE	base	direct
THHLD_NUMADLT_2.0	THHLD_NUMADLT	base	direct
THHLD_NUMADLT_3.0	THHLD_NUMADLT	base	direct
THHLD_NUMKID	THHLD_NUMKID	base	direct
TW_START_1.0	TW_START	base	direct
UI_APPLY_1.0	UI_APPLY	base	direct
WORRY_2.0	WORRY	base	direct
WRKLOSS_1.0	WRKLOSS	base	direct
polit_1	polit	base	direct

In [635…

```
# loop through v1 and v2 map and eliminate non-priority features so
# we don't have both
```

```
find = (df_feats['selected'].isin(['direct']))
updated_pairs = []

for pair in v1_v2_map:
    num_v1s = len(df_feats.loc[find & (df_feats['col_root'].isin(pair['v1']))])
    num_v2s = len(df_feats.loc[find & (df_feats['col_root'].isin(pair['v2']))])

    if num_v1s > 0 and num_v2s > 0:
        remove = 'v2' if pair['priority'] == 'v1' else 'v1'

        df_feats.loc[df_feats['col_root'].isin(pair[remove]),
                     'selected'] = 'removed'
        updated_pairs = updated_pairs + pair['v1'] + pair['v2']

with pd.option_context('display.max_rows', 200):
    display(df_feats.loc[(df_feats['col_root'].isin(updated_pairs)) &
                         (df_feats['selected'].isin(['direct', 'removed']))])
```

	col_root	feat_group	selected
ENROLL1_-1.0	ENROLL1	v1	removed
ENROLL1_1.0	ENROLL1	v1	removed
ENROLL2_-1.0	ENROLL2	v1	removed
ENROLL2_1.0	ENROLL2	v1	removed
ENROLL3_-1.0	ENROLL3	v1	removed
ENROLL3_1.0	ENROLL3	v1	removed
EST_MSA_41860.0	EST_MSA	v1	direct
EST_MSA_42660.0	EST_MSA	v1	direct
HLTHINS1_1.0	HLTHINS1	v1	direct
HLTHINS3_1.0	HLTHINS3	v1	direct
HLTHINS4_-1.0	HLTHINS4	v1	direct
HLTHINS4_1.0	HLTHINS4	v1	direct
HLTHINS5_-1.0	HLTHINS5	v1	direct
HLTHINS6_-1.0	HLTHINS6	v1	direct
HLTHINS7_-1.0	HLTHINS7	v1	direct
HLTHINS8_-1.0	HLTHINS8	v1	direct
HLTHINS_-1.0	HLTHINS	removed	removed
HLTHINS_0.0	HLTHINS	dummy_drop	removed
HLTHINS_1.0	HLTHINS	v2	removed
IN_METRO_AREA_0.0	IN_METRO_AREA	dummy_drop	removed
IN_METRO_AREA_1.0	IN_METRO_AREA	v2	removed
LIVQTR_1.0	LIVQTR	v1	direct
LIVQTR_10.0	LIVQTR	v1	direct
LIVQTR_2_-1.0	LIVQTR_2	removed	removed

	col_root	feat_group	selected
LIVQTR_2_1.0	LIVQTR_2	v2	removed
LIVQTR_2_2.0	LIVQTR_2	dummy_drop	removed
LIVQTR_2_3.0	LIVQTR_2	v2	removed
LIVQTR_2_4.0	LIVQTR_2	v2	removed
MS_2_-1.0	MS_2	removed	removed
MS_2_1.0	MS_2	dummy_drop	removed
MS_2_2.0	MS_2	v2	removed
MS_2_3.0	MS_2	v2	removed
MS_5.0	MS	v1	direct
SCHOOL_KIDS_1.0	SCHOOL_KIDS	v2	direct

Rather than bring in all labels of the category if any label was selected, I'm going to keep only the selected labels for modeling purposes.

In [636…
```
# get final list of features to keep
final_keep = list(df_feats.loc[df_feats['selected']=='direct'].index)
print(len(final_keep))
final_keep
```

```
56
```

Out[636…
```
['AGE',
 'ANXIOUS_2.0',
 'ANYWORK_1.0',
 'CHNGHOW12_1.0',
 'CHNGHOW1_1.0',
 'CHNGHOW2_1.0',
 'CHNGHOW4_1.0',
 'CHNGHOW6_1.0',
 'CHNGHOW8_1.0',
 'CHNGHOW9_1.0',
 'CURFOODSUF',
 'DOWN_3.0',
 'DOWN_4.0',
 'EEDUC',
 'EIP_3.0',
 'EIP_4.0',
 'EST_MSA_41860.0',
 'EST_MSA_42660.0',
 'EXPCTLOSS_1.0',
 'EXPNS_DIF',
 'FEWRTRANS_3.0',
 'FEWRTRIPS_1.0',
 'HADCOVID_1.0',
 'HLTHINS1_1.0',
 'HLTHINS3_1.0',
 'HLTHINS4_-1.0',
 'HLTHINS4_1.0',
 'HLTHINS5_-1.0',
 'HLTHINS6_-1.0',
 'HLTHINS7_-1.0',
 'HLTHINS8_-1.0',
 'INCOME',
 'INTEREST_2.0',
```

```
'LIVQTR_1.0',
'LIVQTR_10.0',
'MS_5.0',
'NOTGET_-1.0',
'PLNDTRIPS_1.0',
'PROP_FOODSPEND_HOME',
'RRACE_3.0',
'RRACE_4.0',
'SCHOOL_KIDS_1.0',
'SNAP_YN_1.0',
'SPNDSRC1_1.0',
'SPNDSRC2_1.0',
'SPNDSRC4_1.0',
'SSA_RECV_1.0',
'TENURE_3.0',
'THHLD_NUMADLT_2.0',
'THHLD_NUMADLT_3.0',
'THHLD_NUMKID',
'TW_START_1.0',
'UI_APPLY_1.0',
'WORRY_2.0',
'WRKLOSS_1.0',
'polit_1']
```

Modeling in Reduced Feature Space

Logistic Regression

In [637…

```
param_grid = {
    'C':[0.01, 1, 100, 1e6],
    'penalty': ['l1', 'l2', 'elasticnet'],
    'solver': ['liblinear', 'saga', 'lbfgs']
}

gs_lr = GridSearchCV(lr, param_grid, scoring='f1_macro', n_jobs=-1,
                     verbose=True)

if rerun_gridsearch:
    gs_lr.fit(X_train_df[final_keep], y_train)

    print(gs_lr.best_estimator_)
    print(gs_lr.best_score_)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   20.3s
[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed:  1.3min finished
LogisticRegression(C=100, class_weight='balanced', max_iter=600)
0.5995781848919467
```

In [640…

```
gs_lr.best_estimator_.get_params
```

Out[640…

```
<bound method BaseEstimator.get_params of LogisticRegression(C=100, class_weight
='balanced', max_iter=600)>
```

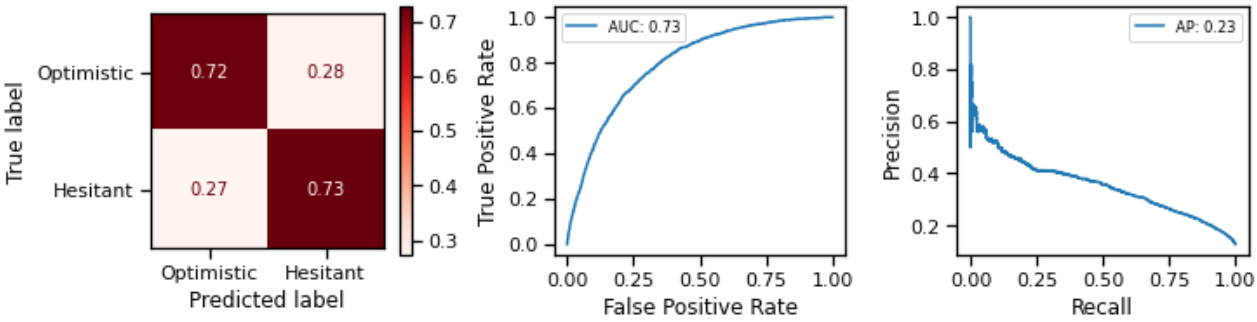
Best LR params on reduced features:

- LogisticRegression(C=100, class_weight='balanced')
  - default solver lbfgs and penalty l2 are implied, since they're defaults
- mean F1 macro: 0.60

In [641…

```
# Logistic Regression with reduced features, optimal params from grid search
lr = LogisticRegression(C=100, class_weight='balanced', max_iter=600,
                        solver='lbfgs', penalty='l2')

lr.fit(X_train_df[final_keep], y_train)

dstools.eval_clf_model(lr, X_test_df[final_keep], y_test, X_train_df[final_keep],
                y_train, labels=labels)
```

```
****************************** Training Data ******************************
              precision    recall  f1-score   support

         0.0       0.95      0.71      0.81     51283
         1.0       0.26      0.74      0.39      7085

    accuracy                           0.72     58368
   macro avg       0.61      0.73      0.60     58368
weighted avg       0.87      0.72      0.76     58368


****************************** Test Data ******************************
              precision    recall  f1-score   support

         0.0       0.95      0.72      0.82     17037
         1.0       0.27      0.73      0.40      2420

    accuracy                           0.72     19457
   macro avg       0.61      0.73      0.61     19457
weighted avg       0.86      0.72      0.77     19457


****************************** Training Scores ******************************
            Training Macro F1 = 0.6003
         Training Macro Recall = 0.725
    Training Balanced Accuracy = 0.725

****************************** Test Scores ******************************
                Test Macro F1 = 0.6082
             Test Macro Recall = 0.7254
        Test Balanced Accuracy = 0.7254

****************************** Differences ******************************
          Train-Test Macro F1 Diff = 0.007900000000000018
       Train-Test Macro Recall Diff = 0.0004000000000006697
Train-Test Balanced Accuracy Diff = 0.0004000000000006697

****************************** Graphs for Test ******************************
```



Not much different from the original model on V1 features that didn't use regularization. So removing the features that the selection process voted were not important hasn't had much effect on Logistic Regression accuracy or F1.

This is OK; it just means I have simplified my model somewhat and honed in on the features that actually do help predictions. I don't think I can go any further with Logistic Regression; let's see how a Random Forest does on the adjusted feature space.

Random Forest

In [638…

```
# grid search with just max_depth and criterion
rf = RandomForestClassifier(class_weight='balanced')

param_grid = {
    'criterion':['gini', 'entropy'],
    'max_depth': [5, 10, 30]
}

gs_rf = GridSearchCV(rf, param_grid, scoring='f1_macro', n_jobs=-1,
                     verbose=True)

if rerun_gridsearch:
    gs_rf.fit(X_train_df[final_keep], y_train)

    print(gs_rf.best_estimator_)
    print(gs_rf.best_score_)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  1.7min finished
RandomForestClassifier(class_weight='balanced', max_depth=10)
0.6181012775387449
```

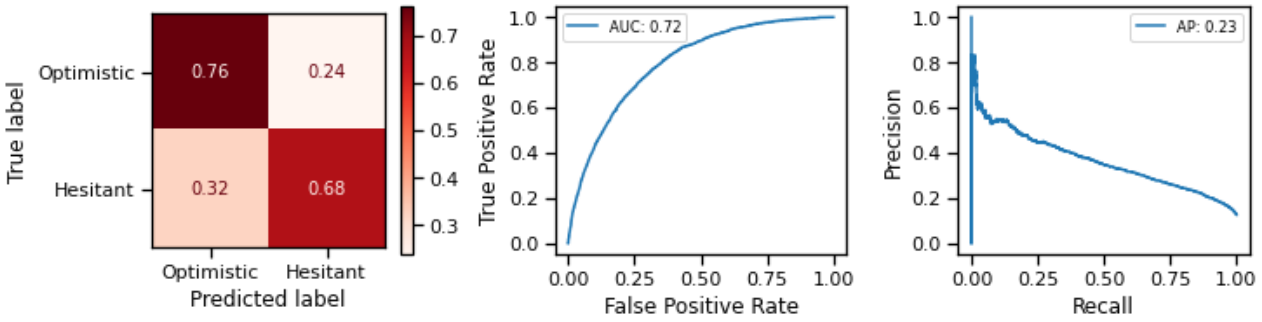
These are the same best params from the previous gridsearch with v1 cols.

In [642…

```
# Random Forest with reduced features, optimal params from grid search
rf = RandomForestClassifier(class_weight='balanced', max_depth=10)

rf.fit(X_train_df[final_keep], y_train)

dstools.eval_clf_model(rf, X_test_df[final_keep], y_test, X_train_df[final_keep]
               y_train, labels=labels)
```

```
****************************** Training Data ******************************
              precision    recall  f1-score   support

         0.0       0.96      0.77      0.86     51283
         1.0       0.32      0.79      0.46      7085

    accuracy                           0.77     58368
   macro avg       0.64      0.78      0.66     58368
weighted avg       0.89      0.77      0.81     58368


****************************** Test Data ******************************
              precision    recall  f1-score   support

         0.0       0.94      0.76      0.84     17037
         1.0       0.29      0.68      0.40      2420

    accuracy                           0.75     19457
   macro avg       0.62      0.72      0.62     19457
weighted avg       0.86      0.75      0.79     19457
```

```
****************************** Training Scores ******************************
            Training Macro F1 = 0.6583
         Training Macro Recall = 0.7799
    Training Balanced Accuracy = 0.7799

****************************** Test Scores ******************************
                Test Macro F1 = 0.6237
             Test Macro Recall = 0.7191
        Test Balanced Accuracy = 0.7191

****************************** Differences ******************************
          Train-Test Macro F1 Diff = -0.034599999999999964
       Train-Test Macro Recall Diff = -0.060800000000000076
Train-Test Balanced Accuracy Diff = -0.060800000000000076

****************************** Graphs for Test ******************************
```



Random Forest on the reduced features with best params from gridsearch yields f1 macro and accuracy, and performance on test data very similar to what I saw in my initial model on v1 cols.

I don't think I can tune further to get better performance, but the reduced features seem to provide just as accurate a result.

In [643…
```
# Saving final models out
joblib.dump(lr, 'final_lr_model.joblib')
joblib.dump(rf, 'final_rf_model.joblib')
```

Out[643…
```
['final_rf_model.joblib']
```

iNTERPRET

I used the coefficients from the logistic regression model, since they allowed me to determine interpretable odds for predictors of both hesitancy and optimisim.

I did also initially review the feature importances and permutation importances from the random forest model. However, only some of these agreed with each other, and with the logistic regression coefficients.

I reviewed which coefficients 2 out of 3 methods agreed on being in the top 30, and considered dropping those logistic regression coefficients that the random forest model didn't agree were as important. However, the random forest model's importances did not seem especially stable, and only `AGE` and `EEDUC` stood out from others, which were both predictors of optimism according to logistic regression. I ultimately decided to make my conclusions and recommendations based solely on the logistic regression coefficients, because I think this

model type is less prone to making somewhat arbitrary decisions based on randomness, and performed the most consistently throughout the modeling process.

See the Extra section at the end of this notebook for the random forest importance work.

In [644…
```
# Load in models, if needed
#lr = joblib.load('final_lr_model.joblib')
#rf = joblib.load('final_rf_model.joblib')
```

Logistic Regression Coefficients

In [647…
```
# convert logit coeffients from log odds to odds ratios
lr_odds = np.exp(lr.coef_[0])
lr_odds_s = pd.Series(lr_odds, index=final_keep, )
lr_odds_s.sort_values(ascending=False)[:10]
```

Out[647…
```
LIVQTR_10.0       2.479889
THHLD_NUMKID      2.286665
EXPNS_DIF         2.244045
CURFOODSUF        1.795916
LIVQTR_1.0        1.728915
FEWRTRANS_3.0     1.509160
RRACE_4.0         1.365280
HADCOVID_1.0      1.320822
polit_1           1.272780
HLTHINS4_-1.0     1.258194
dtype: float64
```

In [674…
```
# Odds greater than 1. Higher values represent greater odds of being vaccine
# hesitant, my target class of 1
np.round(lr_odds_s.loc[lr_odds_s >= 1]\
         .sort_values(ascending=False).head(10),2)
```

Out[674…
```
LIVQTR_10.0       2.48
THHLD_NUMKID      2.29
EXPNS_DIF         2.24
CURFOODSUF        1.80
LIVQTR_1.0        1.73
FEWRTRANS_3.0     1.51
RRACE_4.0         1.37
HADCOVID_1.0      1.32
polit_1           1.27
HLTHINS4_-1.0     1.26
dtype: float64
```

The top predictors of vaccine hesitancy increased odds over 50% are:

Model Column Name	Description of Question	Odds of Hesitancy
LIVQTR_10.0	Household residence is a boat, RV, or van	2.48
THHLD_NUMKID	Number of individuals under 18 in the household	2.29
EXPNS_DIF	Level of difficulty meeting household expenses in the past 7 days	2.24
CURFOODSUF	Level of household food insufficiency in the past 7 days	1.80
LIVQTR_1.0	Household residence is a mobile home	1.73

Model Column Name	Description of Question	Odds of Hesitancy
FEWRTRANS_3.0	Respondent did not use public transportation such as bus, rail, or ride-share before the pandemic, so transportation did not change	1.51

In [673…
```
# Odds less than 1. All odds here are related to being in the target class,
# so odds less than 1 are more likely to be in the 0 class.

# Using 1 / odds to convert to odds that respondent is in 0 class, or
# vaccine optimistic
np.round(1 / (lr_odds_s.loc[lr_odds_s < 1])\
         .sort_values(ascending=True).head(10), 2)
```

Out[673…
```
AGE                6.06
EEDUC              2.62
RRACE_3.0          2.22
EST_MSA_41860.0    2.18
CHNGHOW6_1.0       2.03
INCOME             1.89
FEWRTRIPS_1.0      1.88
TW_START_1.0       1.51
ANXIOUS_2.0        1.38
HLTHINS3_1.0       1.34
dtype: float64
```

The predictors of vaccine optimism with increased odds over 50% are:

Model Column Name	Description of Question	Odds of Optimism
AGE	Respondent's age in years	6.06
EEDUC	Level of education	2.62
RRACE_3.0	Respondent identified as Asian	2.22
EST_MSA_41860.0	Household is in the San Francisco-Oakland-Berkeley, CA Metro Area	2.18
CHNGHOW6_1.0	Members of the household had avoided eating at restaurants in the prior 7 days	2.03
INCOME	Pre-tax income level	1.89
FEWRTRIPS_1.0	Members of the household had taken fewer trips to stores because of the pandemic in the prior 7 days	1.88
TW_START_1.0	At least one adult in the household substituted some or all of their typical in-person work for telework	1.51

Since the household being in San Francisco isn't very applicable to the rest of the country, I'm going to leave it out of my recommendations.

Interpretation Visualizations

In [980…
```
# Why Not reason columns and column mapping

hes_reason_cols = ['WHYNOT1', 'WHYNOT2', 'WHYNOT3', 'WHYNOT4', 'WHYNOT5',
                   'WHYNOT6', 'WHYNOT7', 'WHYNOT8', 'WHYNOT9', 'WHYNOT10',
                   'WHYNOT11']

hes_reasonb_cols = ['WHYNOTB1', 'WHYNOTB2', 'WHYNOTB3', 'WHYNOTB4', 'WHYNOTB5',
```

```
                  'WHYNOTB6']

reasons_map = {
'WHYNOT1' : "Concerned about side effects",
'WHYNOT2' : "Don't know if it will work",
'WHYNOT3' : "Don't believe I need it",
'WHYNOT4' : "Don't like vaccines",
'WHYNOT5' : "Doctor has not recommended it",
'WHYNOT6' : "Plan to wait and see if it is safe -- may get it later",
'WHYNOT7' : "Other people need it more than I do right now",
'WHYNOT8' : "Concerned about the cost",
'WHYNOT9' : "Don't trust these vaccines",
'WHYNOT10' : "Don't trust the government",
'WHYNOT11' : "Other",
'WHYNOTB1' : "Already had COVID",
'WHYNOTB2' : "Not in a high risk group",
'WHYNOTB3' : "Plan to use masks and other precautions instead",
'WHYNOTB4' : "Don't believe COVID is a serious illness",
'WHYNOTB5' : "Don't think vaccines are beneficial",
'WHYNOTB6' : "Other"
}
```

In [836…

```
# create separate hesitant and optimistic columns for easy grouping
df.loc[df['target']==1, 'Hesitant'] = 1
df.loc[df['target']==0, 'Optimistic']= 1

df.loc[df['target']==1, 'target_str'] = 'Hesitant'
df.loc[df['target']==0, 'target_str']= 'Optimistic'
```

In [750…

```
# make initial df of hesitant respondents
hes_df = df.loc[(df['target']==1)].copy()

# remove the -88 and -99 values to leave only the responses
for col in hes_reason_cols + hes_reasonb_cols:
    hes_df.loc[hes_df[col].isin([-88, -99]), col] = np.nan

# drop rows where no why questions were answered
hes_df.dropna(axis=0, how='all', subset=hes_reason_cols + hes_reasonb_cols,
              inplace=True)
hes_df[hes_reason_cols + hes_reasonb_cols]
```

Out[750…

	WHYNOT1	WHYNOT2	WHYNOT3	WHYNOT4	WHYNOT5	WHYNOT6	WHYNOT7	WHYNO
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
5	NaN	NaN	NaN	NaN	NaN	1.0	NaN	N
7	1.0	1.0	1.0	NaN	NaN	NaN	1.0	N
21	NaN	NaN	1.0	NaN	NaN	NaN	NaN	N
24	1.0	NaN	NaN	NaN	NaN	NaN	NaN	N
...	...	...	...	...	...	...	...	
77767	1.0	1.0	NaN	NaN	NaN	NaN	NaN	N
77786	1.0	NaN	NaN	NaN	NaN	NaN	NaN	N
77788	1.0	NaN	NaN	NaN	NaN	1.0	NaN	N
77812	NaN	NaN	NaN	1.0	NaN	NaN	NaN	N

	WHYNOT1	WHYNOT2	WHYNOT3	WHYNOT4	WHYNOT5	WHYNOT6	WHYNOT7	WHYNC
77816	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

9477 rows × 17 columns

In [116…

```
def percent_cattarget_per_catpred(df, pred_col,
                                  target_cols=['Hesitant', 'Optimistic']):
    """Creates a dataframe suitable for plotting the proportion of each
    label in a categorical predictor that falls into each target class.

    ***
    Args:

    df: type = Dataframe
        Dataframe passed should include `pred_col` and `target_cols`. Any
        other columns will be ignored.
        Note that while for modeling target is typically represented in a
        single column, for this function each target class should be dummied
        into its own column, so that the number of observations in each class
        can be easily aggregated using the count() function.

    pred_col: type = string
        Column name of the categorical predictor in `df`.

    target_cols: type: list
        List of dummied target column names where each column represents a
        target class.

    ***
    Returns a dataframe where each row represents the proportion of
    observations grouped by `pred_col` and `target_cols`. Such a dataframe
    may be easily turned into a Seaborn bar graph that shows how the
    proportion oftarget class changes for each categorical predictor label,
    using `x="pred_col", y="percent", hue="target", data=df`.


    `pred_col`: using the original predictor column name
        Populated with the predictor labels.

    `target`:
        Populated with target classes taken from the dummied column names.

    `percent`:
        Proportion of observations that fell into predictor and target class
        represented by that row.
    """

    # get total count for each category of the predictor
    totals = df[pred_col].value_counts()

    # group by predictor and populate a column per target category with the
    # percentage of target cat in that pred category
    per = df.groupby(pred_col)[target_cols].count()\
     .apply(lambda x: x / totals.loc[x.name], axis=1)

    per.reset_index(inplace=True)
```

```
    melted = pd.melt(per, id_vars=pred_col, var_name="target",
                     value_name="percent")
    return melted
```

In [115…

```
def top_n_whys_perpred(filt_df, pred_col, pred_vals, top_n=5,
                       why_cols=hes_reason_cols,
                       mapper=reasons_map):

    # get total count of respondents in the filtered df, per pred
    # to use as percentage denominator
    if pred_col == None:
        total = len(filt_df)
    else:
        totals = filt_df[pred_col].value_counts()

    # get a list of the top n reasons people who belonged to pred_vals group
    # selected why they were hesitant
    if pred_vals == None:
        top_5_whys = list(filt_df[why_cols].sum()\
                          .sort_values(ascending=False)[:top_n].index)
    else:
        top_5_whys = list(filt_df.loc[filt_df[pred_col].isin(pred_vals),
                    why_cols].sum().sort_values(ascending=False)[:top_n].index)

    # group by predictor and populate a column per why reason with
    # percentage respondents who selected that reason
    # note that the reasons were multi-select, so respondents may have selected
    # more than one and they will probably not add to 100
    # also we are only choosing the top n most poplar out of 11 based on
    # most popular across all filtered pred values
    if pred_col == None:
        final = pd.DataFrame(filt_df[top_5_whys].count()\
                             .apply(lambda x: x / total), columns=['percent'])
        final.reset_index(inplace=True)

        final.rename(columns={'index':'target'}, inplace=True)

        # map reasons to column names
        final['target'].replace(mapper, inplace=True)


    else:
        filt_df = filt_df.groupby(pred_col)[top_5_whys].count()\
                  .apply(lambda x: x / totals.loc[x.name], axis=1)

        filt_df.reset_index(inplace=True)

        # map reasons to column names
        filt_df.rename(mapper=mapper, axis=1, inplace=True)

        final = pd.melt(filt_df, id_vars=pred_col, var_name="target",
                        value_name="percent")

    return final
```

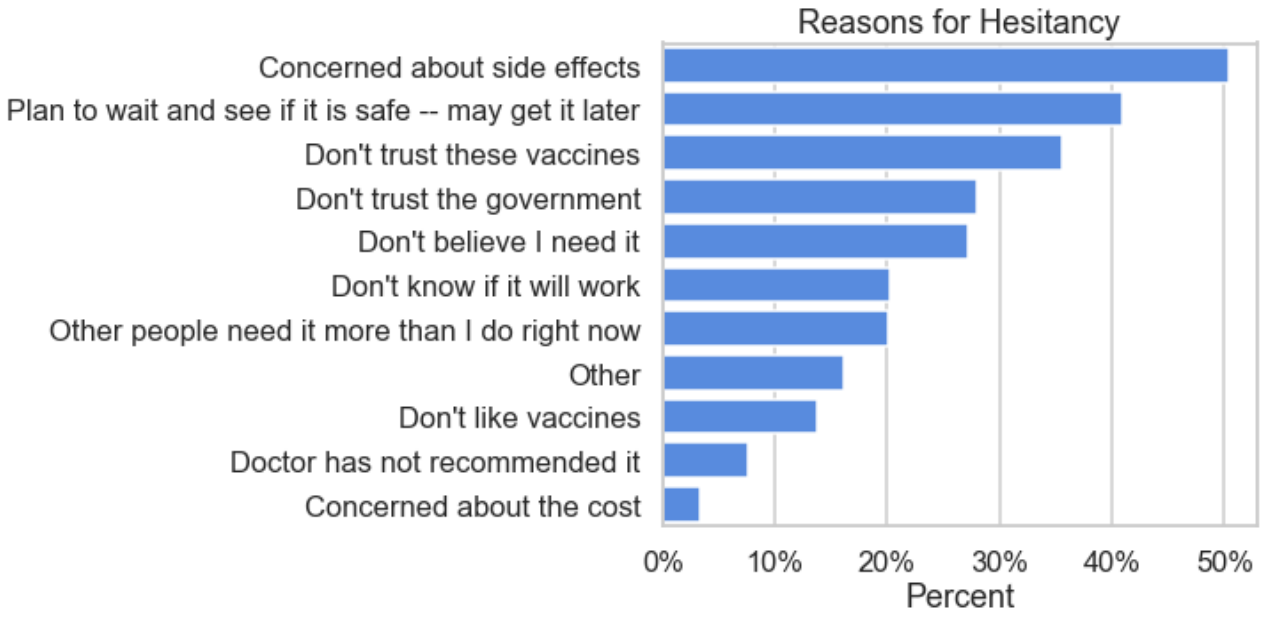
In [122…

```
colors = sns.color_palette(['#4285f4', '#34a853'])
sns.set_style("whitegrid")
sns.set_context('talk')
```

Reasons for Hesitancy

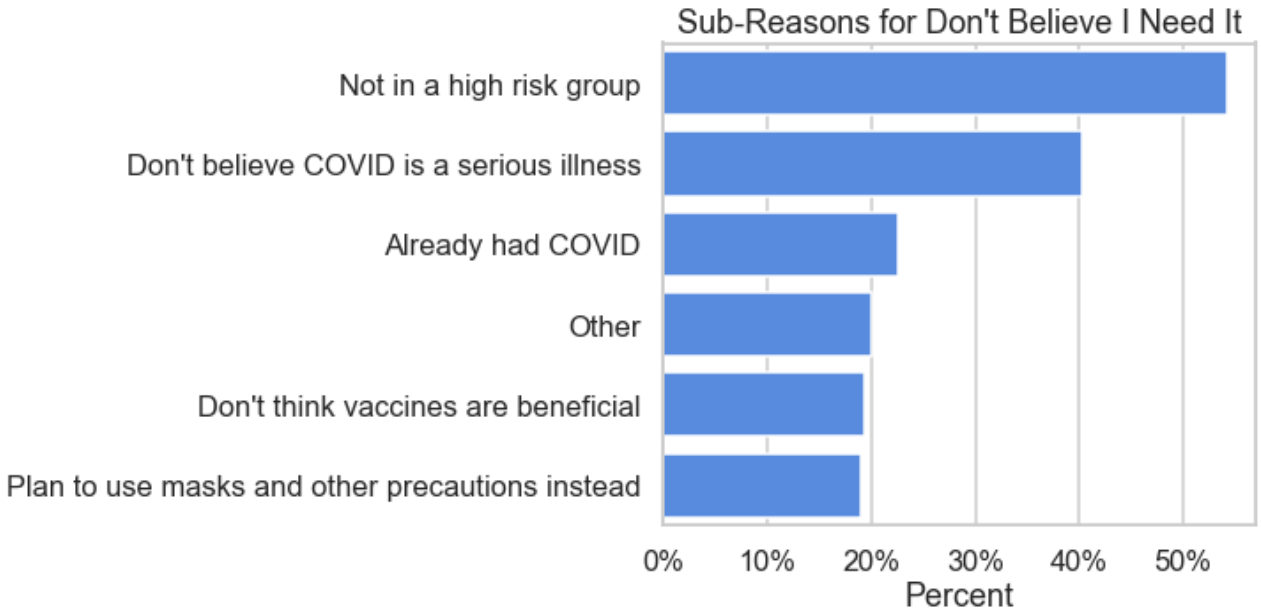
In [124…

```
# Plot reasons for hesitancy reported in descending order based on
# percentage
pred = None
pred_pretty = 'Reasons for Hesitancy'

to_plot = top_n_whys_perpred(hes_df, pred, None, top_n=11)

fig, ax = plt.subplots(figsize=(6, 5))
sns.barplot(y='target', x="percent", data=to_plot, ax=ax, orient='h',
            color='#4285f4')


ax.set_title(f'Reasons for Hesitancy')
ax.set_xlabel('Percent')
ax.set_ylabel(None)
ax.xaxis.set_major_formatter(ticker.PercentFormatter(xmax=1));
```



In [124…

```
# Plot subreasons for not beleiving they need it
pred = None
pred_pretty = 'Reasons for Hesitancy'

# get df filtered by people who were shown the subreasons because they said
# they didn't beleive they needed it

to_plot = top_n_whys_perpred(hes_df.loc[hes_df['WHYNOT3']==1], pred, None,
                             why_cols=hes_reasonb_cols, top_n=6)

fig, ax = plt.subplots(figsize=(6, 5))
sns.barplot(y='target', x="percent", data=to_plot, ax=ax, orient='h',
            color='#4285f4')


ax.set_title(f"Sub-Reasons for Don't Believe I Need It")
ax.set_xlabel('Percent')
ax.set_ylabel(None)
ax.xaxis.set_major_formatter(ticker.PercentFormatter(xmax=1));
```



In [126…

```
# how many people chose both 'concerned about side effects' and 'wait and see'
# versus just one or the other?

# side effects only
only_side_effects = len(hes_df.loc[(hes_df['WHYNOT1']==1) & (hes_df['WHYNOT6'].i
only_waitsee = len(hes_df.loc[(hes_df['WHYNOT6']==1) & (hes_df['WHYNOT1'].isna()
both_effects_waitsee = len(hes_df.loc[(hes_df['WHYNOT6']==1) & (hes_df['WHYNOT1'
one_or_other = len(hes_df.loc[(hes_df['WHYNOT6']==1) | (hes_df['WHYNOT1']==1)])

print(f"Concerned about side effects only: {only_side_effects/one_or_other}")
print(f"Plan to wait and see only: {only_waitsee/one_or_other}")
print(f"Concerned about side effects AND will wait and see: {both_effects_waitse
```

```
Concerned about side effects only: 0.3808158062460166
Plan to wait and see only: 0.23932441045251754
Concerned about side effects AND will wait and see: 0.3798597833014659
```

Hesitancy - Type of Housing

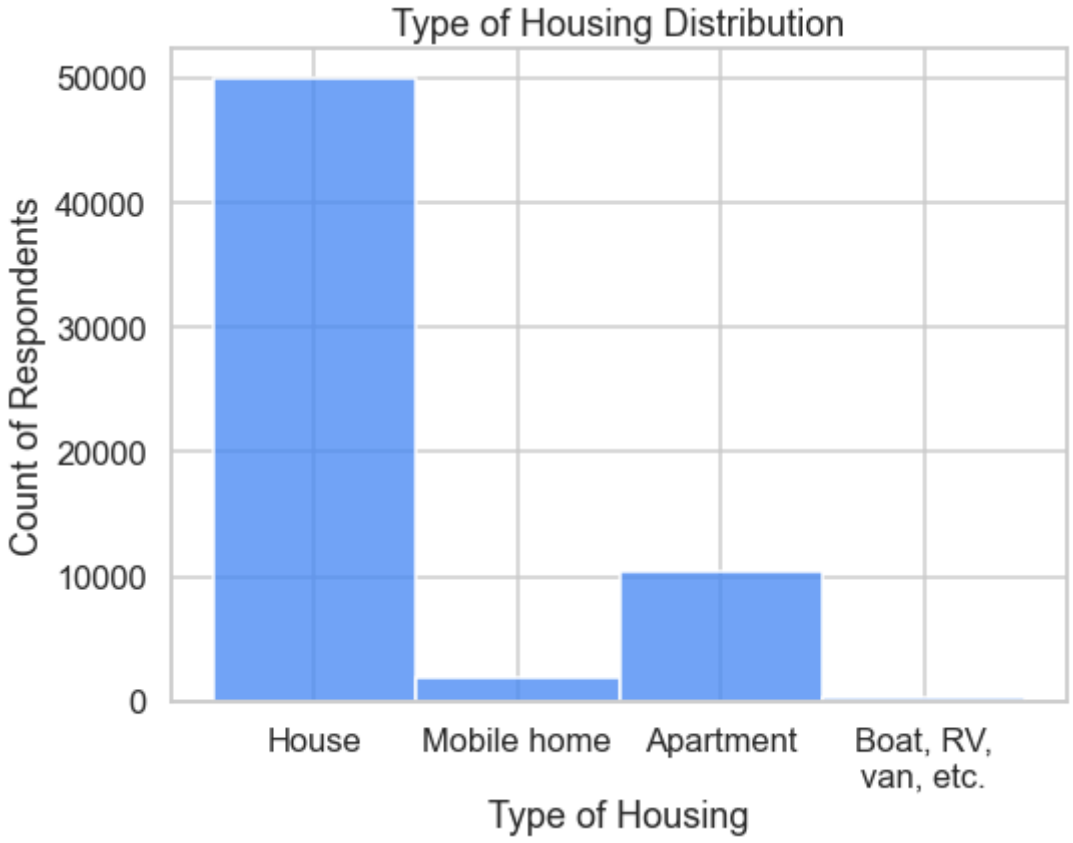
In [111…

```
# mapper for LIVQTR_2
livqtr_map = {
    1.0:"Mobile home",
    2.0:"House",
    3.0:"Apartment",
    4.0:"Boat, RV,\nvan, etc."
}
```

In [127…

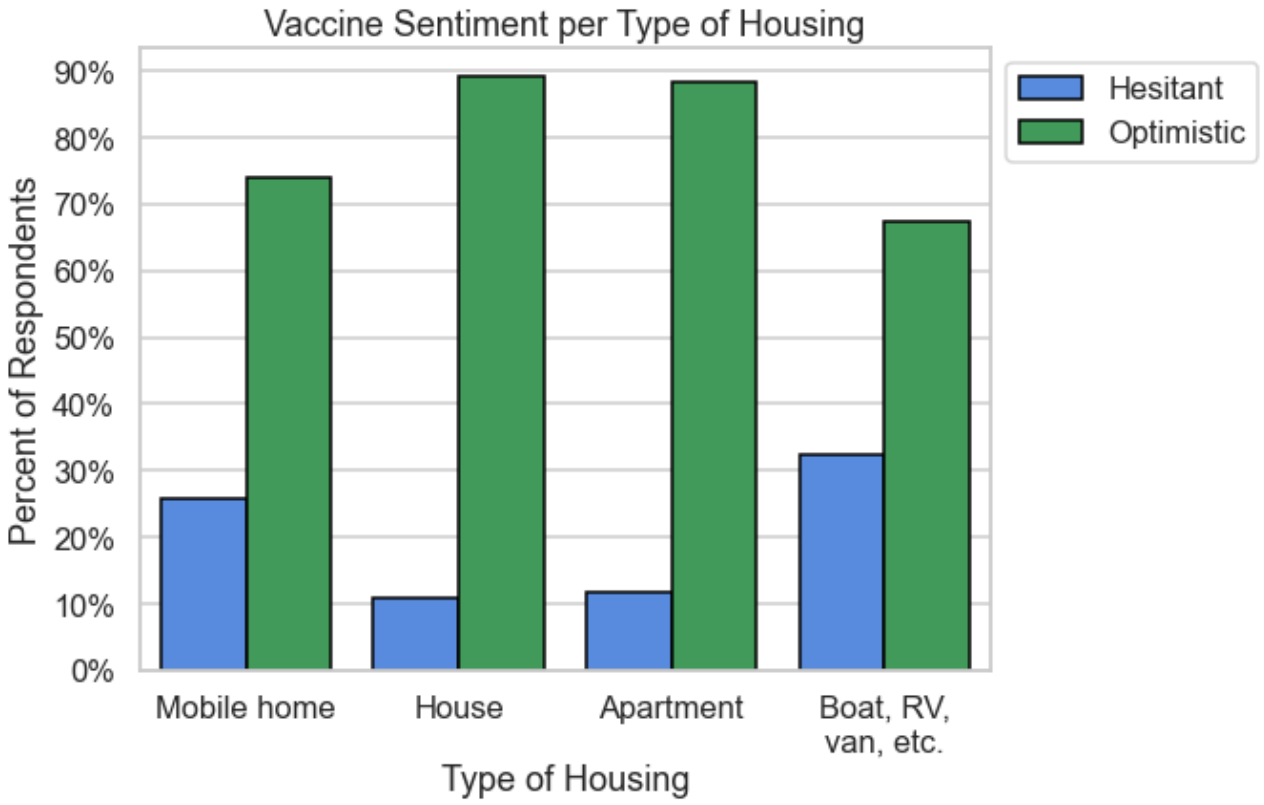
```
# visualize distribution of people in each housing category

pred = 'LIVQTR_2'
pred_pretty = 'Type of Housing'

filter_df = df.loc[df[pred] > 0].copy()

# map placholder values to text
filter_df[pred].replace(livqtr_map, inplace=True)

fig, ax = plt.subplots(figsize=(8, 6))
sns.histplot(x=filter_df[pred], color='#4285f4')
```

```
ax.set_title(f'{pred_pretty} Distribution')
ax.set_xlabel(pred_pretty)
ax.set_ylabel('Count of Respondents');
```



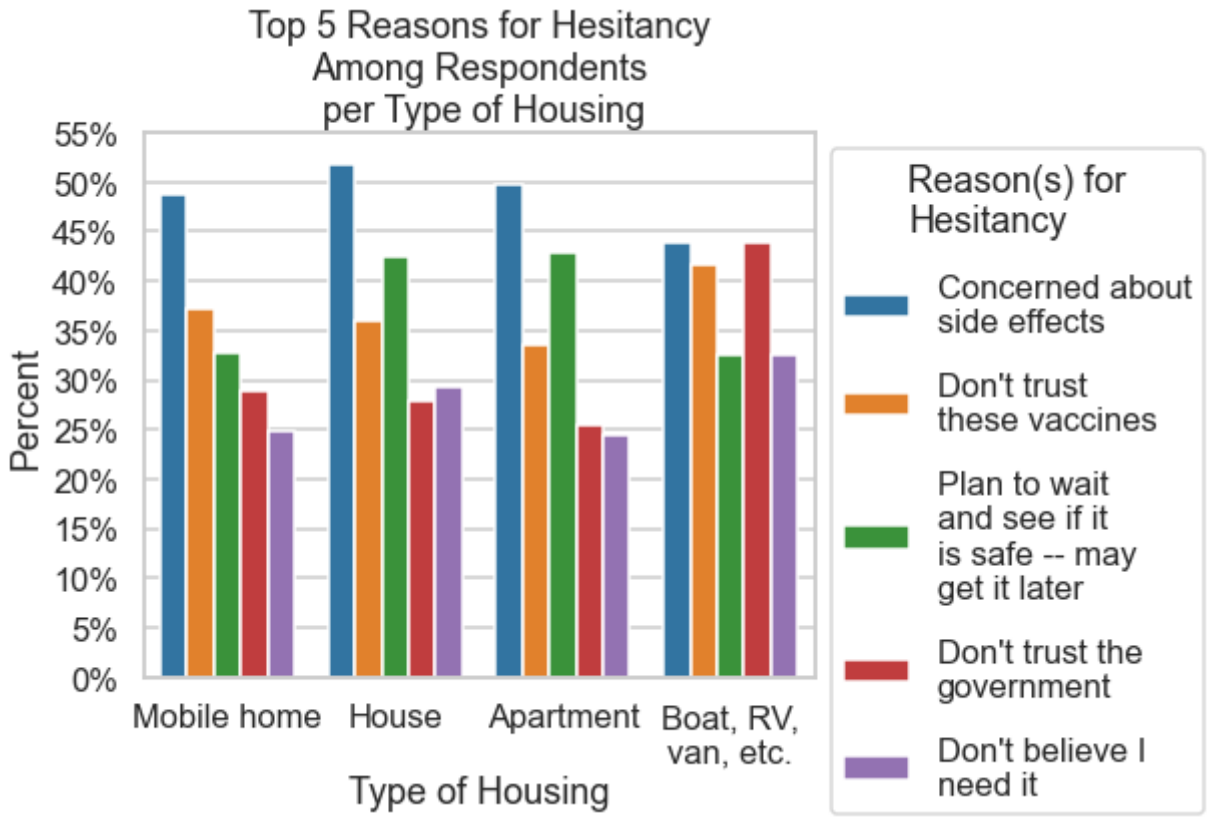
In [127…

```
# Plot sentiment for number of Housing Type

filter_df = df.loc[df[pred] > 0]
to_plot = percent_cattarget_per_catpred(filter_df, pred)

# map placholder values to text
to_plot[pred].replace(livqtr_map, inplace=True)

fig, ax = plt.subplots(figsize=(8, 6))
sns.barplot(x=pred, y="percent", hue="target",\
                data=to_plot, palette=colors, ax=ax, edgecolor='black')


ax.set_title(f'Vaccine Sentiment per {pred_pretty}')
ax.set_xlabel(pred_pretty)
ax.set_ylabel('Percent of Respondents')
ax.legend(title=None, bbox_to_anchor=(1, 1), loc='upper left')
#ax.tick_params(axis='x', labelrotation = 45)
ax.yaxis.set_ticks(np.arange(0, 1, 0.1))
ax.yaxis.set_major_formatter(ticker.PercentFormatter(xmax=1));
```



In [125…

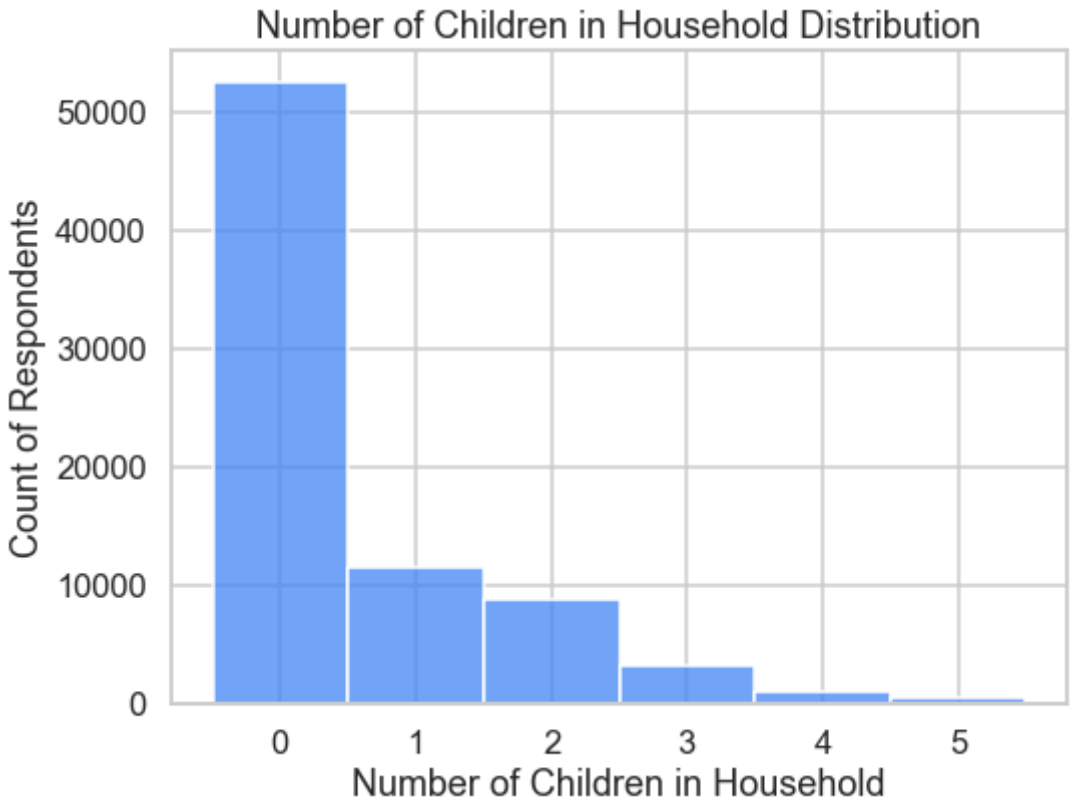
```
# plot top 5 reasons respondents said why they didn't plan to get vaccinated
filt_df = hes_df.loc[hes_df[pred] > 0, [pred] + hes_reason_cols]

to_plot = top_n_whys_perpred(filt_df, pred, [1.0, 4.0])
# map placholder values to text
to_plot[pred].replace(livqtr_map, inplace=True)

fig, ax = plt.subplots(figsize=(6, 5))
sns.barplot(x=pred, y="percent", hue="target",\
                data=to_plot, ax=ax, orient='v')

# nifty wrap labels code from:
#https://stackoverflow.com/questions/47057789/matplotlib-wrap-text-in-legend
handles, labels = ax.get_legend_handles_labels()
labels = ['\n'.join(wrap(l, 15)) for l in labels]

ax.set_title(f'Top 5 Reasons for Hesitancy\nAmong Respondents\n per Type of Hous
ax.set_xlabel(pred_pretty)
ax.set_ylabel('Percent')
ax.legend(handles=handles, labels=labels, title="Reason(s) for\nHesitancy", labe
          bbox_to_anchor=(1, 1), loc='upper left')
ax.yaxis.set_ticks(np.arange(0, 0.6, 0.05))

#ax.xaxis.set_major_formatter(ticker.FormatStrFormatter("#%d"))
ax.yaxis.set_major_formatter(ticker.PercentFormatter(xmax=1));
```



Hesitancy - Number of Children

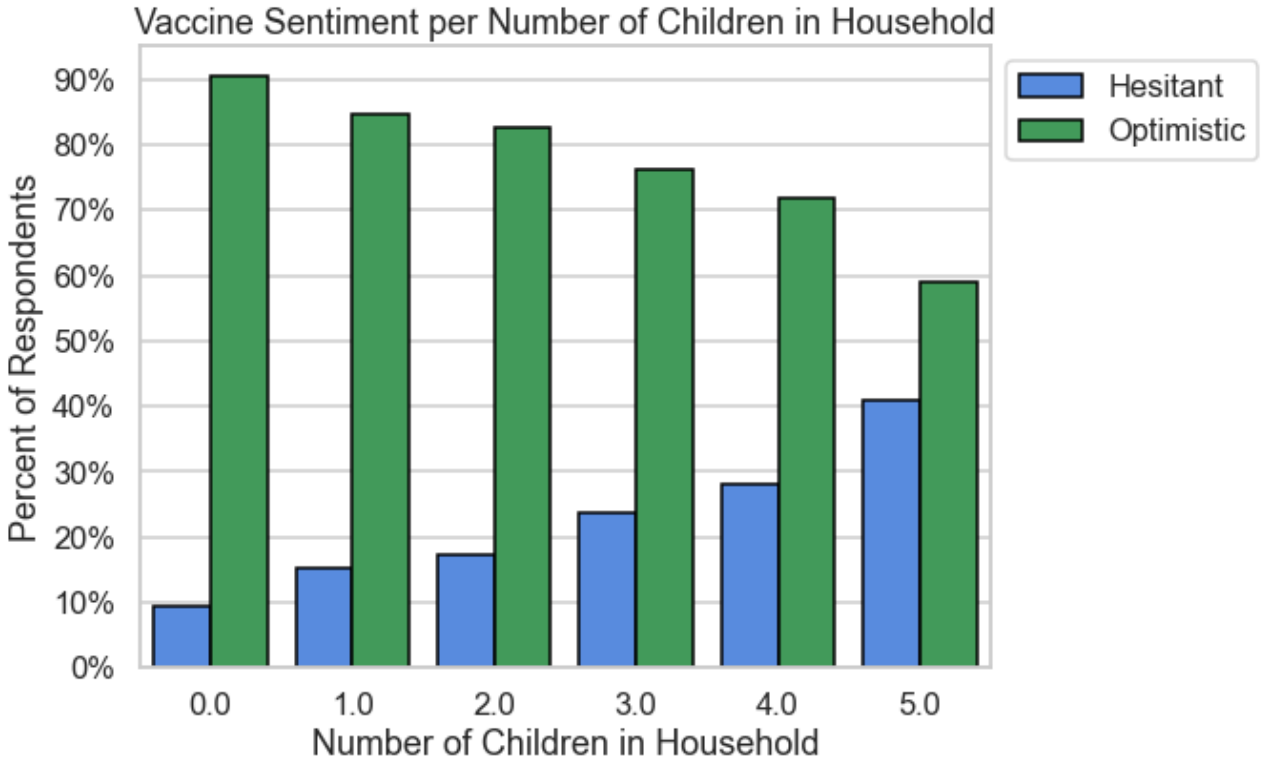
In [128…

```
# visualize distribution of number of children

pred = 'THHLD_NUMKID'
pred_pretty = 'Number of Children in Household'

fig, ax = plt.subplots(figsize=(8, 6))
sns.histplot(x=df[pred], color='#4285f4', discrete=True)

ax.set_title(f'{pred_pretty} Distribution')
ax.set_xlabel(pred_pretty)
ax.set_ylabel('Count of Respondents');
```



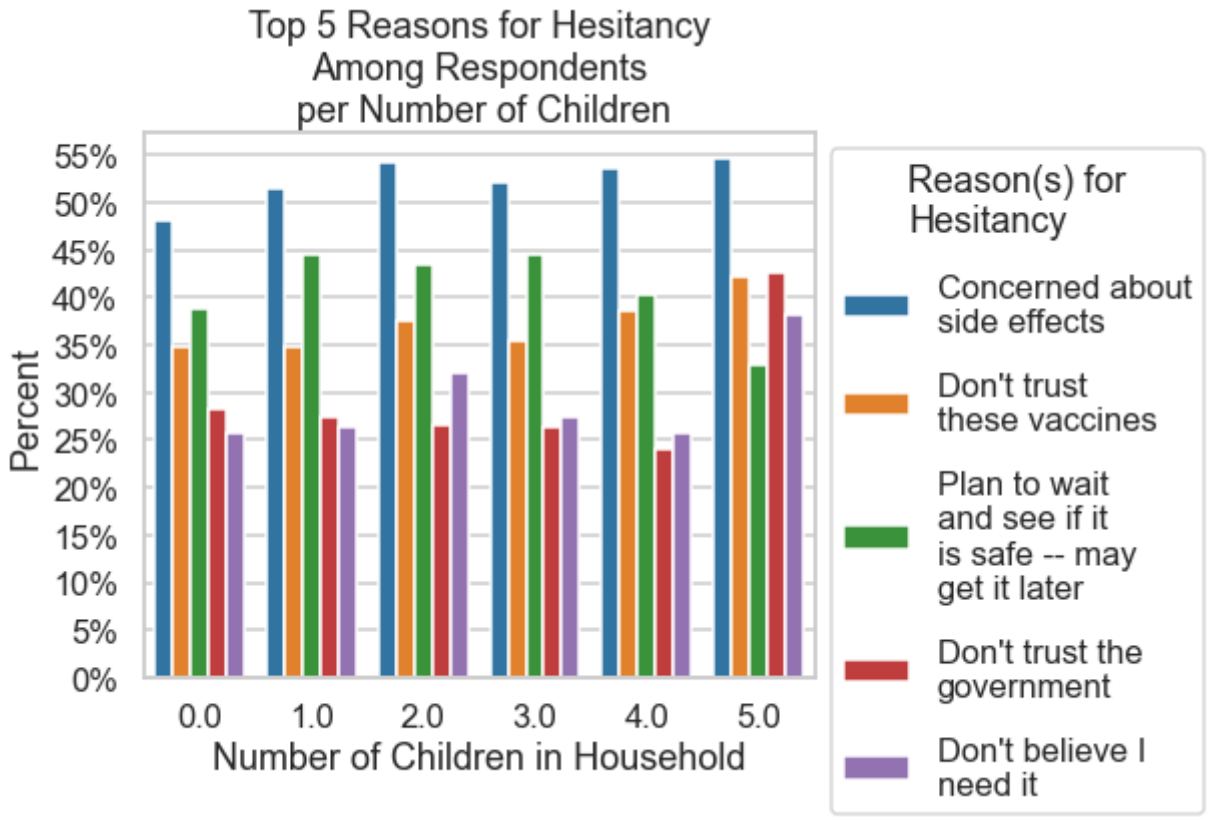
In [125…

```
# Plot sentiment for number of children in the household

to_plot = percent_cattarget_per_catpred(df, pred)

fig, ax = plt.subplots(figsize=(8, 6))
sns.barplot(x=pred, y="percent", hue="target",\
                data=to_plot, palette=colors, ax=ax, edgecolor='black')
ax.set_title(f'Vaccine Sentiment per {pred_pretty}')
ax.set_xlabel(pred_pretty)
ax.set_ylabel('Percent of Respondents')
ax.legend(title=None, bbox_to_anchor=(1, 1), loc='upper left')
ax.yaxis.set_ticks(np.arange(0, 1, 0.1))
ax.yaxis.set_major_formatter(ticker.PercentFormatter(xmax=1));
```



In [125…

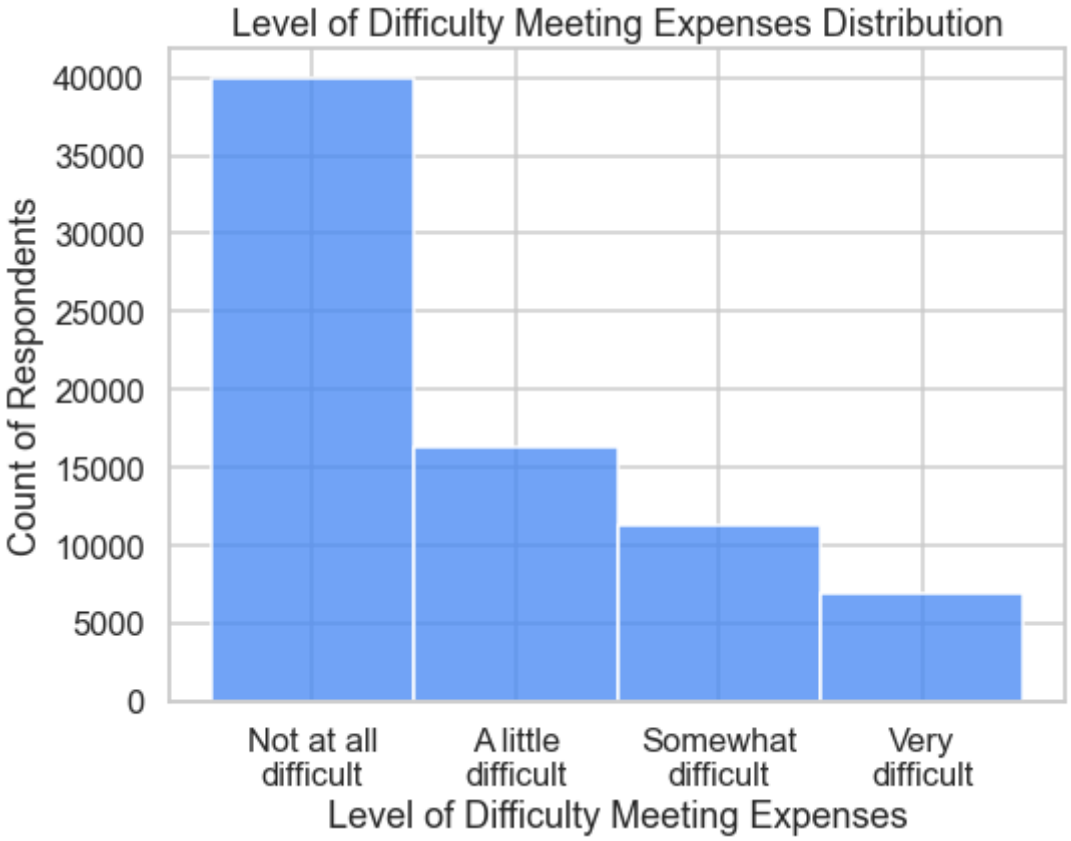
```
# plot top 5 reasons respondents said why they didn't plan to get vaccinated
#filt_df = hes_df.loc[hes_df[pred] > 0, [pred] + hes_reason_cols]

to_plot = top_n_whys_perpred(hes_df, pred, [4.0, 5.0])

fig, ax = plt.subplots(figsize=(6, 5))
sns.barplot(x=pred, y="percent", hue="target",\
                data=to_plot, ax=ax, orient='v')

# nifty wrap labels code from:
#https://stackoverflow.com/questions/47057789/matplotlib-wrap-text-in-legend
handles, labels = ax.get_legend_handles_labels()
labels = ['\n'.join(wrap(l, 15)) for l in labels]

ax.set_title(f'Top 5 Reasons for Hesitancy\nAmong Respondents\n per Number of Ch
ax.set_xlabel(pred_pretty)
ax.set_ylabel('Percent')
ax.legend(handles=handles, labels=labels, title="Reason(s) for\nHesitancy", labe
          bbox_to_anchor=(1, 1), loc='upper left')
ax.yaxis.set_ticks(np.arange(0, 0.6, 0.05))

#ax.xaxis.set_major_formatter(ticker.FormatStrFormatter("#%d"))
ax.yaxis.set_major_formatter(ticker.PercentFormatter(xmax=1));
```



Hesitancy - Difficulty Meeting Household Expenses

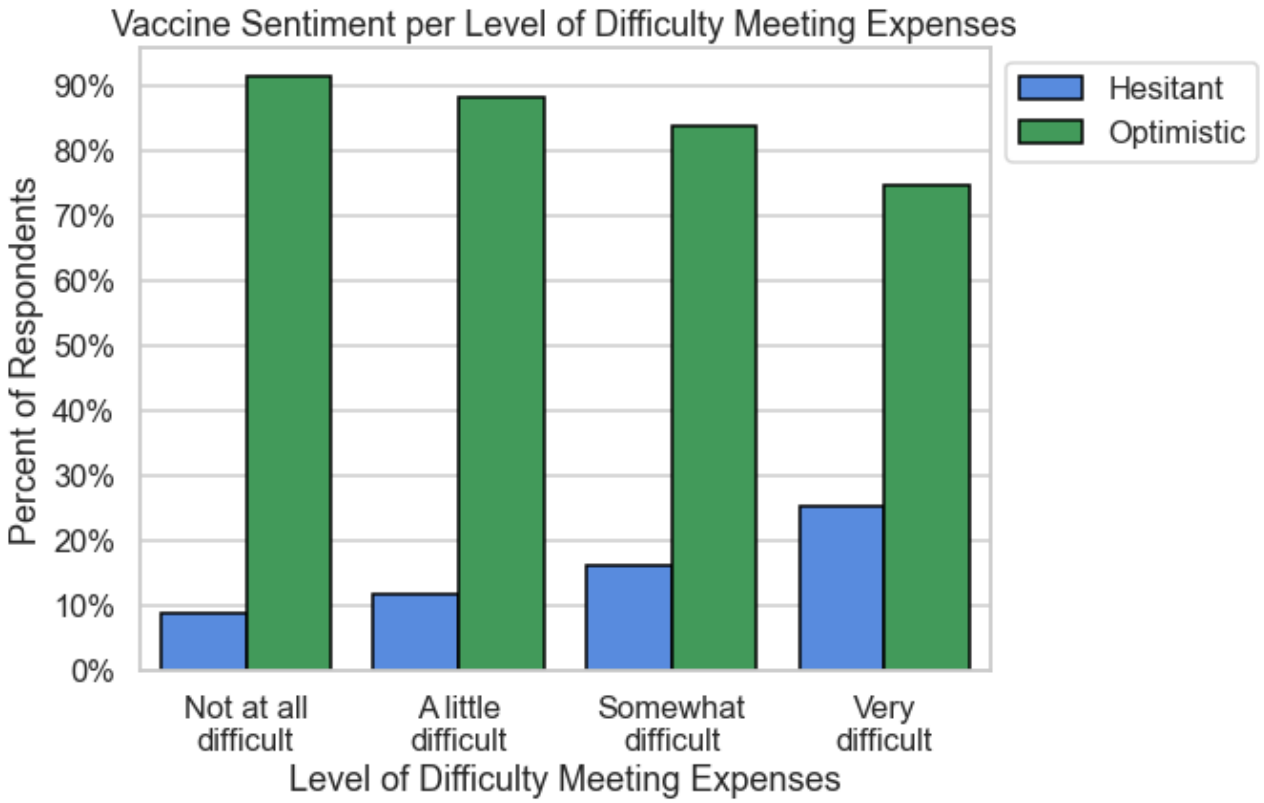
In [130…

```
# visualize distribution of expense difficulty

# mapper for EXPNS_DIF
expns_map = {
    1.0:"Not at all\ndifficult",
    2.0:"A little\ndifficult",
    3.0:"Somewhat\ndifficult",
    4.0:"Very\ndifficult"
}

pred = 'EXPNS_DIF'
pred_pretty = 'Level of Difficulty Meeting Expenses'

filter_df = df.loc[df[pred]>0, pred].copy()
filter_df.replace(expns_map, inplace=True)
filter_df = filter_df.astype('category')
filter_df.cat.set_categories(list(expns_map.values()), ordered=True,
                             inplace=True)


fig, ax = plt.subplots(figsize=(8, 6))
sns.histplot(x=filter_df, color='#4285f4', discrete=True)

ax.set_title(f'{pred_pretty} Distribution')
ax.set_xlabel(pred_pretty)
ax.set_ylabel('Count of Respondents');
```



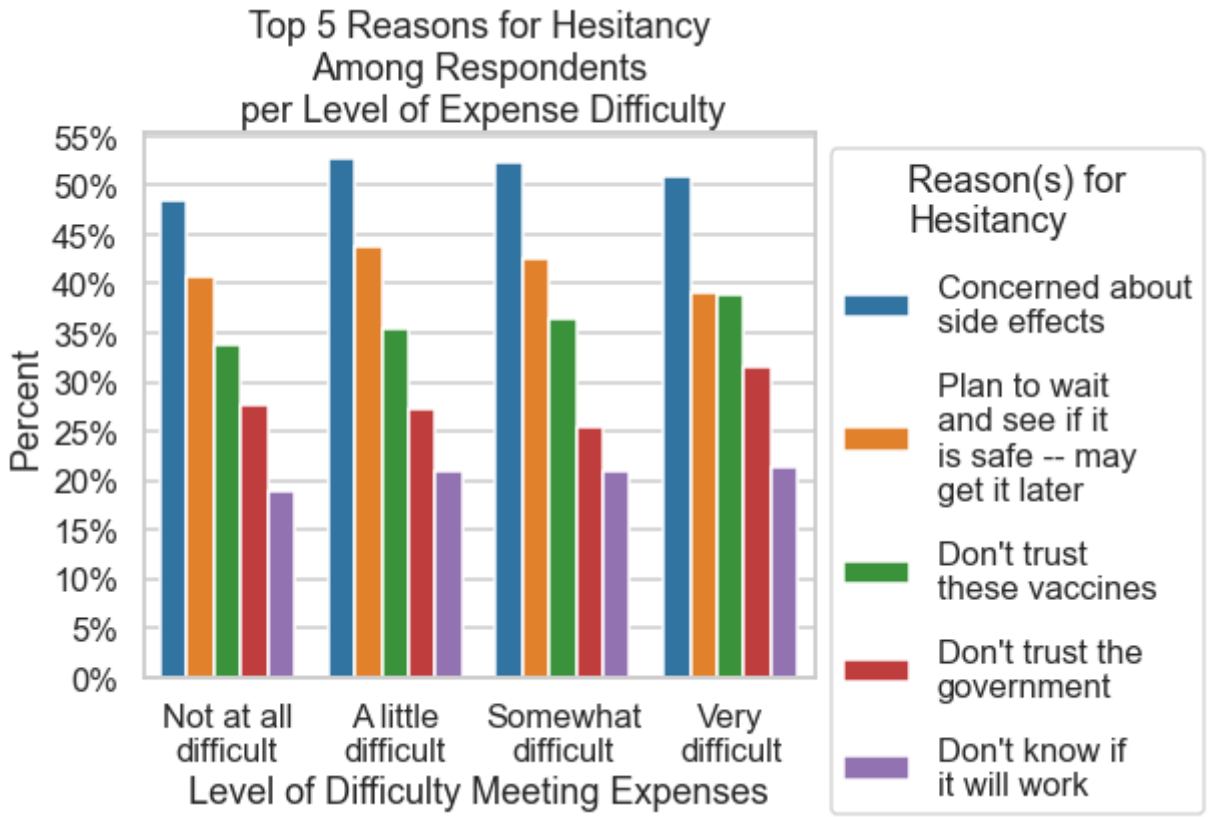
In [125…

```
# Plot sentiment for difficulty meeting expenses


filter_df = df.loc[df[pred] > 0]
to_plot = percent_cattarget_per_catpred(filter_df, pred)

# map placholder values to text
to_plot[pred].replace(expns_map, inplace=True)

fig, ax = plt.subplots(figsize=(8, 6))
sns.barplot(x=pred, y="percent", hue="target",\
                data=to_plot, palette=colors, ax=ax, edgecolor='black')


ax.set_title(f'Vaccine Sentiment per {pred_pretty}')
ax.set_xlabel(pred_pretty)
ax.set_ylabel('Percent of Respondents')
ax.legend(title=None, bbox_to_anchor=(1, 1), loc='upper left')
#ax.tick_params(axis='x', labelrotation = 45)
ax.yaxis.set_ticks(np.arange(0, 1, 0.1))
ax.yaxis.set_major_formatter(ticker.PercentFormatter(xmax=1));
```



In [126…

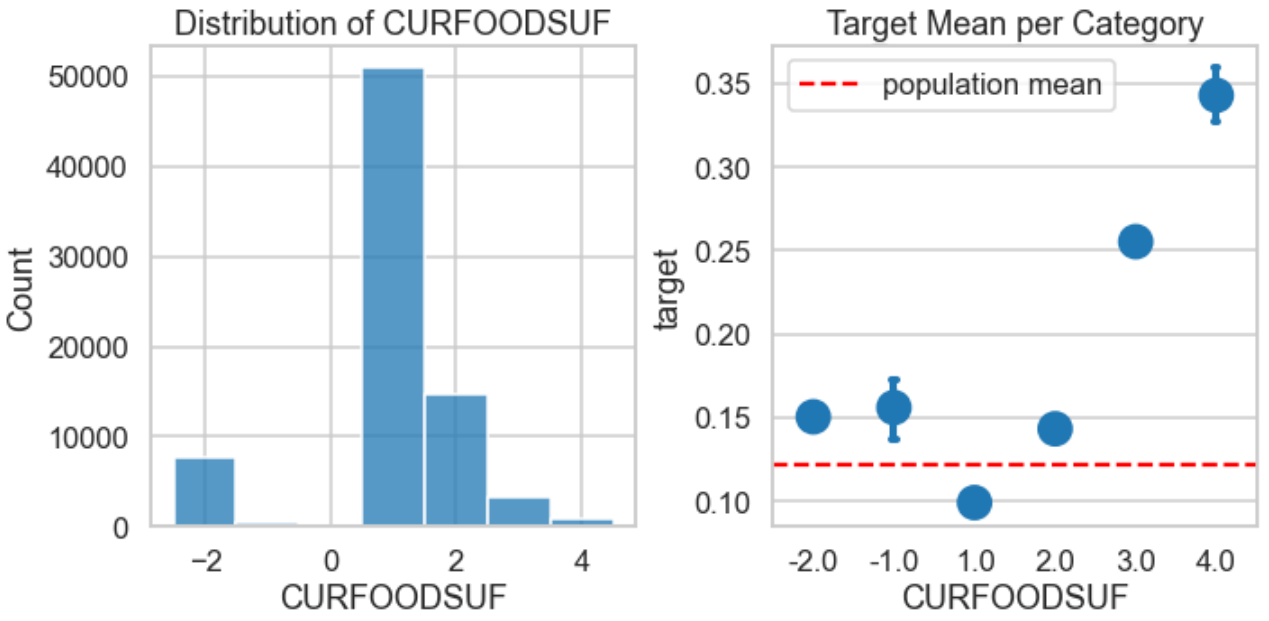
```
# plot top 5 reasons respondents said why they didn't plan to get vaccinated
filt_df = hes_df.loc[hes_df[pred] > 0, [pred] + hes_reason_cols]

to_plot = top_n_whys_perpred(filt_df, pred, [4.0])
# map placholder values to text
to_plot[pred].replace(expns_map, inplace=True)

fig, ax = plt.subplots(figsize=(6, 5))
sns.barplot(x=pred, y="percent", hue="target",\
                data=to_plot, ax=ax, orient='v')

# nifty wrap labels code from:
#https://stackoverflow.com/questions/47057789/matplotlib-wrap-text-in-legend
handles, labels = ax.get_legend_handles_labels()
labels = ['\n'.join(wrap(l, 15)) for l in labels]

ax.set_title(f'Top 5 Reasons for Hesitancy\nAmong Respondents\n per Level of Exp
ax.set_xlabel(pred_pretty)
ax.set_ylabel('Percent')
ax.legend(handles=handles, labels=labels, title="Reason(s) for\nHesitancy",
          labelspacing=1,
          bbox_to_anchor=(1, 1), loc='upper left')
ax.yaxis.set_ticks(np.arange(0, 0.6, 0.05))

#ax.xaxis.set_major_formatter(ticker.FormatStrFormatter("#%d"))
ax.yaxis.set_major_formatter(ticker.PercentFormatter(xmax=1));
```



Hesitancy - Food Insufficiency

In [122…

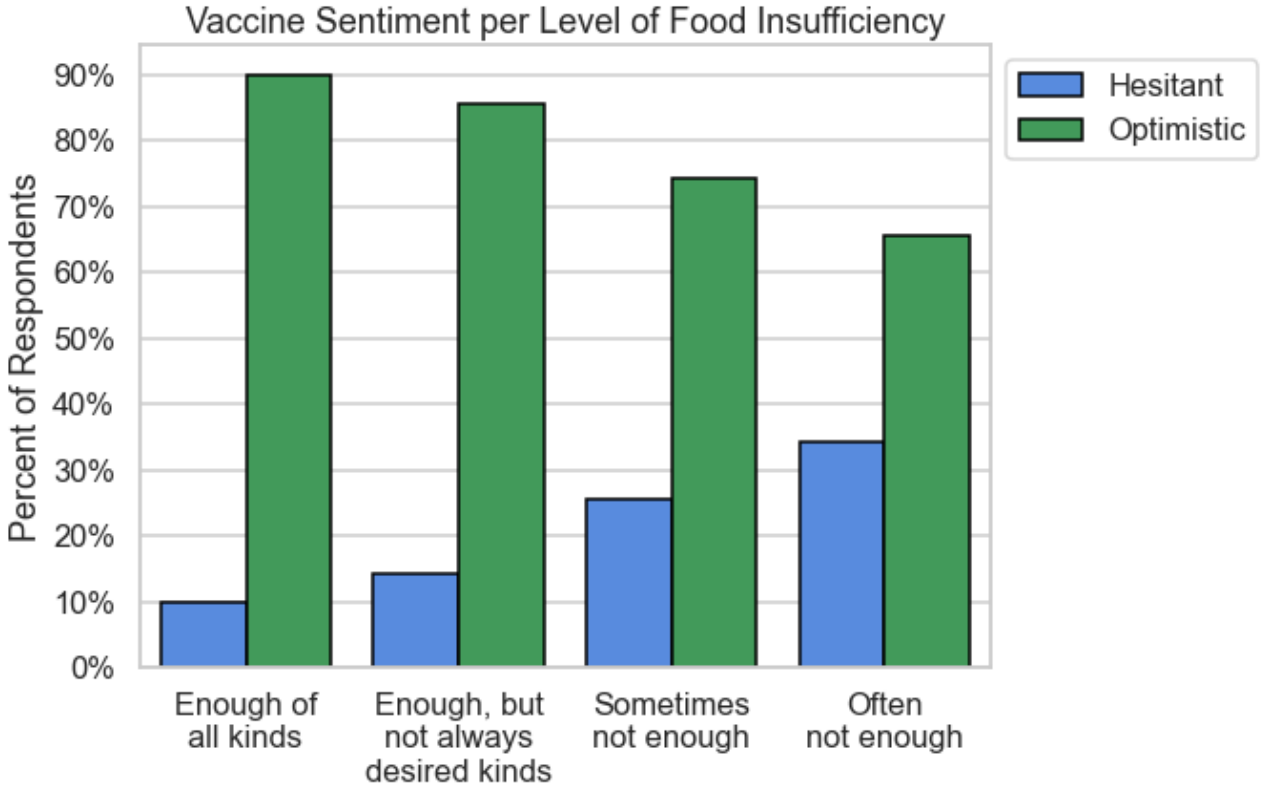
```
dstools.explore_data_catbin(['CURFOODSUF'], df, 'target')
```



In [125…

```
# Plot sentiment for difficulty meeting expenses

# mapper for CURFOODSUF
food_map = {
    1.0:"Enough of\nall kinds",
    2.0:"Enough, but\nnot always\ndesired kinds",
    3.0:"Sometimes\nnot enough",
    4.0:"Often\nnot enough"
```

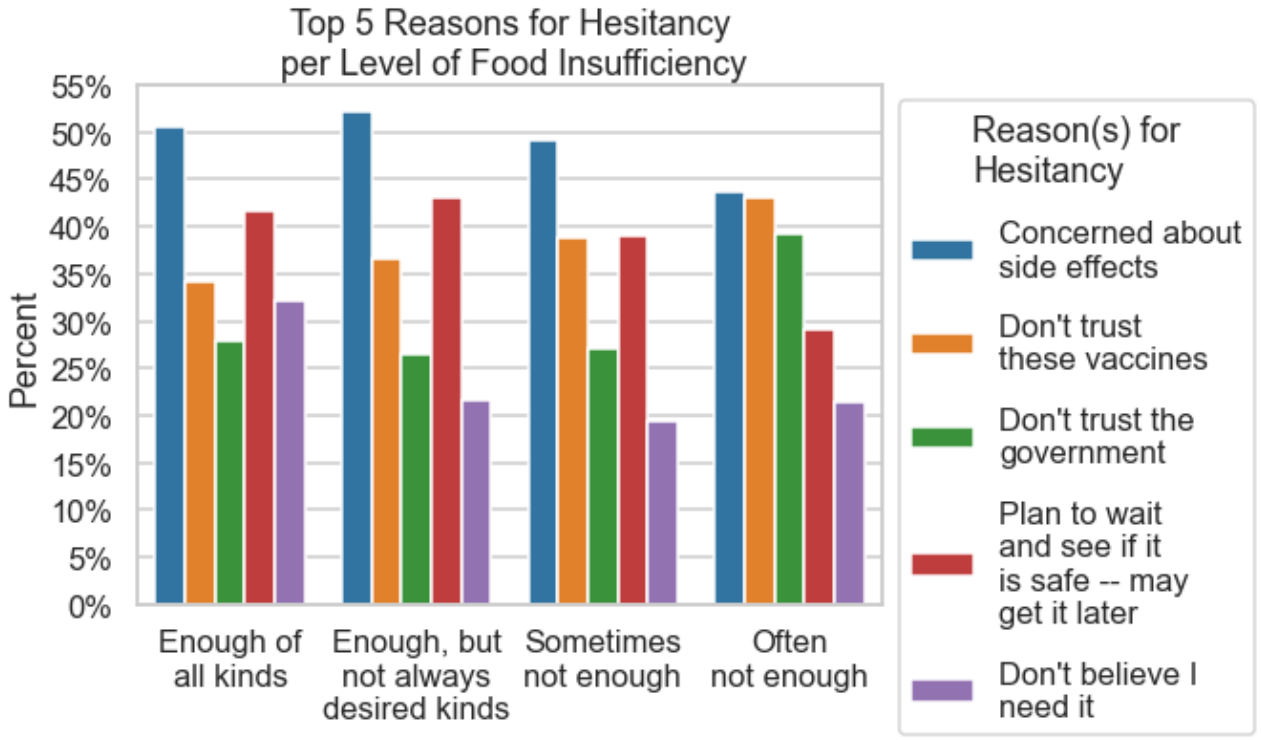
```
}

pred = 'CURFOODSUF'
pred_pretty = 'Level of Food Insufficiency'

filter_df = df.loc[df[pred] > 0]
to_plot = percent_cattarget_per_catpred(filter_df, pred)

# map placholder values to text
to_plot[pred].replace(food_map, inplace=True)

fig, ax = plt.subplots(figsize=(8, 6))
sns.barplot(x=pred, y="percent", hue="target",\
                data=to_plot, palette=colors, ax=ax, edgecolor='black')


ax.set_title(f'Vaccine Sentiment per {pred_pretty}')
ax.set_xlabel(None)
ax.set_ylabel('Percent of Respondents')
ax.legend(title=None, bbox_to_anchor=(1, 1), loc='upper left')
#ax.tick_params(axis='x', labelrotation = 45)
ax.yaxis.set_ticks(np.arange(0, 1, 0.1))
ax.yaxis.set_major_formatter(ticker.PercentFormatter(xmax=1));
```



In [125…

```
# plot top 5 reasons respondents said why they didn't plan to get vaccinated
filt_df = hes_df.loc[hes_df[pred] > 0, [pred] + hes_reason_cols]

to_plot = top_n_whys_perpred(filt_df, pred, [4.0])
# map placholder values to text
to_plot[pred].replace(food_map, inplace=True)

fig, ax = plt.subplots(figsize=(7, 5))
sns.barplot(x=pred, y="percent", hue="target",\
                data=to_plot, ax=ax, orient='v')
```

```
# nifty wrap labels code from:
#https://stackoverflow.com/questions/47057789/matplotlib-wrap-text-in-legend
handles, labels = ax.get_legend_handles_labels()
labels = ['\n'.join(wrap(l, 15)) for l in labels]

ax.set_title(f'Top 5 Reasons for Hesitancy\n per Level of Food Insufficiency')
ax.set_xlabel(None)
ax.set_ylabel('Percent')
ax.legend(handles=handles, labels=labels, title="Reason(s) for\nHesitancy",
          labelspacing=1,
          bbox_to_anchor=(1, 1), loc='upper left')
ax.yaxis.set_ticks(np.arange(0, 0.6, 0.05))

#ax.xaxis.set_major_formatter(ticker.FormatStrFormatter("#%d"))
ax.yaxis.set_major_formatter(ticker.PercentFormatter(xmax=1));
```



Optimism - Age

In [118…

```
hes_meanage = df.loc[df['target']==1, 'AGE'].mean()
hes_medianage = df.loc[df['target']==1, 'AGE'].median()
opt_meanage = df.loc[df['target']==0, 'AGE'].mean()
opt_medianage = df.loc[df['target']==0, 'AGE'].median()

print(f"Mean hesitant age: {hes_meanage}")
print(f"Median hesitant age: {hes_medianage}")
print(f"Mean optimistic age: {opt_meanage}")
print(f"Median optimistic age: {opt_medianage}")
```

```
Mean hesitant age: 47.279221462388215
Median hesitant age: 46.0
Mean optimistic age: 55.57539519906323
Median optimistic age: 57.0
```

In [118…

```
# are the sample distributions normal enough (or large enough) to perform a t-te
stat, pval = stats.normaltest(df.loc[df['target']==1, 'AGE'])
```

```
print(f"Hesitant stat is {stat}")
print(f"Hesitant p-value is {pval}")
```

```
Hesitant stat is 267.84274050988984
Hesitant p-value is 6.8974393339571815e-59
```

In [118…

```
stat, pval = stats.normaltest(df.loc[df['target']==0, 'AGE'])
print(f"Optimistic stat is {stat}")
print(f"Optimistic p-value is {pval}")
```

```
Optimistic stat is 6938.611023197798
Optimistic p-value is 0.0
```

Both samples passed the normality test, so we can do a t-test to see if the differences in their means is statistically significant.

In [118…

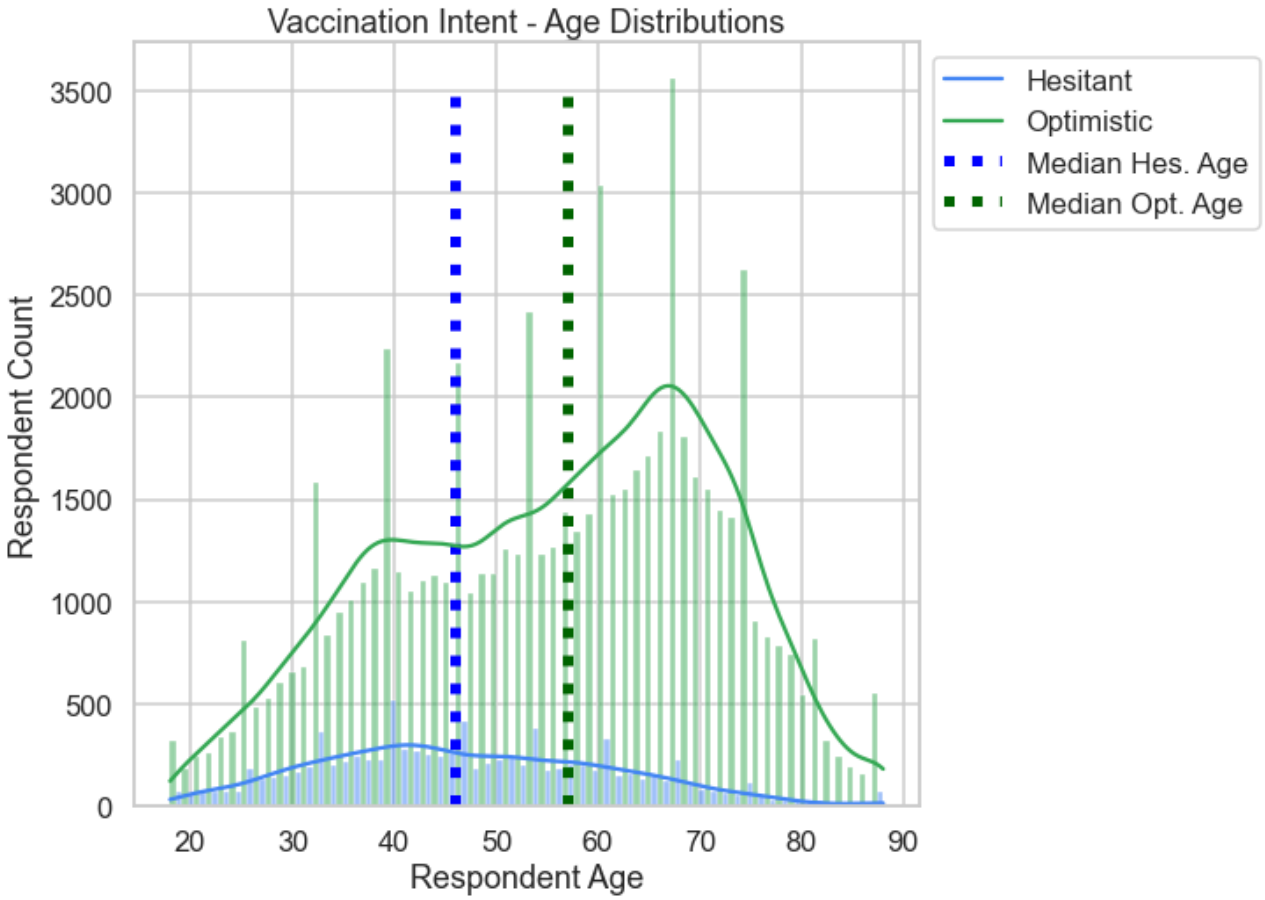
```
# Is the difference in means statistically significant?
# will use welch's t-test since sample size is so different
t, pval = stats.ttest_ind(df.loc[df['target']==1, 'AGE'],
                          df.loc[df['target']==0, 'AGE'])
print(f"t-statistic is {t}")
print(f"p-value is {pval}")
```

```
t-statistic is -48.377312249394244
p-value is 0.0
```

The p-value is very low and t-stat is very low, so the difference is statistically significant.

In [119…

```
# Plot age distributions by vaccine sentiment
hes_meanage = df.loc[df['target']==1, 'AGE'].mean()

colors = sns.color_palette(['#34a853', '#4285f4'])

fig, ax = plt.subplots(figsize=(8, 8))

sns.histplot(x='AGE', data=df, hue='target', ax=ax, palette=colors,
             kde=True, multiple='dodge')

ax.set_title('Vaccination Intent - Age Distributions')
ax.set_ylabel('Respondent Count')
ax.set_xlabel('Respondent Age')


ax.vlines(hes_medianage, ymin=0, ymax=3500,
          color='blue', linestyles='dotted', linewidth=6)
ax.vlines(opt_medianage, ymin=0, ymax=3500,
          color='darkgreen', linestyles='dotted', linewidth=6)
ax.legend(labels=['Hesitant', 'Optimistic', 'Median Hes. Age',
                  'Median Opt. Age'], bbox_to_anchor=(1, 1), loc='upper left');
```



CONCLUSIONS & RECOMMENDATIONS

Understanding the Hesitant Population

I see no clear pattern from the top predictors of hesitancy that might be used in targeted outreach programs.

Although characteristics such as living in certain types of dwellings and having more children in the household indicate a higher likelihood of hesitancy, households with no children living with them and who live in houses and apartments are still hesitant at about the same percentage as the entire population.

For characteristics that predicted a greater percentage of hesitant respondents, it was the rarest categories that had the highest percentage of hesitancy. It's possible that being in a group that could be considered in some sense on the fringe--in other words living under circumstances which are the least common among the population--is a sort of meta-predictor of hesitancy.

Overall, my conclusion based on this analysis is that there is no one single characteristic that hesitant people have in common; their situations are varied.

It's worth noting that the predictors I attempted to engineer to represent political leanings (household being in a "Red" versus "Blue" state, or being in a metropolitan area, which tend to

be Blue) didn't make it into the top predictors of hesitancy or optimism. It's possible we would see different results if we had data on the specific household's political affiliations, but this data did not support the idea that vaccine hesitancy is largely a partisan issue.

Reasons for Hesitancy

The top 5 reasons cited for hesitancy across the hesitant group were:

1. Concerned about side effects
2. Plan to wait and see if it's safe -- may get it later
3. Don't trust the vaccines
4. Don't trust the government
5. Don't believe I need it

Of people who said they didn't believe they needed the vaccine, top sub-reasons why were:

1. Not in a high risk group
2. Don't believe COVID is a serious illness

When we look at the top 5 reasons in the groups with the highest percentage of hesitancy, there is almost no difference compared to the groups with average percentage of hesitancy. The only group that differed was households who said it had been very difficult to meet household expenses, where the number 5 reason was "Don't believe it will work".

Approaches that address the top 5 hesitancy reasons should produce results across the board.

Recommendations for Further Initiatives to Encourage Vaccination

Address the concerns about side effects

Since concerns about side effects are a top reason cited for hesitancy, the government should try to ease these concerns.

Currently, the top result of Googling "COVID vaccine side effects" is this CDC webpage. Although factually accurate, and I believe genuinely designed to educate the public on what to expect, seeing so many warnings can be scary.

If we had more data on the likelihood of experiencing moderate to severe side effects, I think that would help put things in perspective. Moderate to severe side effects may be less common than people assume based on what they hear anecdotally, so statistics to understand their prevalence.

Carry on with current campaigns offering perks and cash

The next most common reason was waiting to see if it was safe. This survey was conducted in early March, so by the time of this writing in late May, some people may already have been

convinced. However, the existing campaigns offering money or perks such as food coupons may help sway the hesitant sooner, especially since they tended to be groups with lower income, greater difficulty meeting expenses, and greater food insufficiency.

Also, since only 61% of adults have been vaccinated at this point (compared to the 82% of people who said they were optimistic in the last survey from May 10) it's clear there are still people who actually do intend to get vaccinated, but just need the right motivation to get it done sooner. The existing campaigns may help convince people who are optimistic, but have held off for other reasons.

Potential Future Analysis

The Household Pulse Surveys are still ongoing, and there is already more recent microdata available than there was when this analysis was run.

Continuing to evaluate the current machine learning models on new data may help validate the accuracy of these results.

It's also very possible that vaccine sentiment and reasons for hesitancy will shift over time as more people become vaccinated and it becomes more commonplace. Understanding how people's opinion changes over time may be useful for future public health endeavors, not only the current pandemic.

Extra

Random Forest Feature Importances

In [650…
```
rf_im = pd.Series(rf.feature_importances_, index=final_keep)
rf_im.sort_values(ascending=False).head(20)
```

Out[650…
```
AGE                     0.148433
EEDUC                   0.092729
CHNGHOW6_1.0            0.081980
EXPNS_DIF               0.061443
INCOME                  0.047147
FEWRTRIPS_1.0           0.045253
THHLD_NUMKID            0.040950
TW_START_1.0            0.037096
CHNGHOW1_1.0            0.035354
HLTHINS3_1.0            0.034170
CURFOODSUF              0.024513
PROP_FOODSPEND_HOME     0.024182
SSA_RECV_1.0            0.023916
WRKLOSS_1.0             0.017526
polit_1                 0.015579
FEWRTRANS_3.0           0.013863
CHNGHOW4_1.0            0.013642
CHNGHOW12_1.0           0.012586
SCHOOL_KIDS_1.0         0.010690
ANXIOUS_2.0             0.010510
dtype: float64
```

Random Forest Permutation Importance

In [651…
```
rf_pi = permutation_importance(rf, X_test_df[final_keep], y_test,
                               scoring='f1_macro', n_jobs=-1, n_repeats=15)
rf_pi.keys()
```

Out[651…
```
dict_keys(['importances_mean', 'importances_std', 'importances'])
```

In [848…
```
rf_pi_s = pd.Series(rf_pi['importances_mean'], index=final_keep)

rf_pi_s.sort_values(ascending=False).head(30)
```

Out[848…
```
AGE                   0.025719
EEDUC                 0.010079
TW_START_1.0          0.007177
CHNGHOW6_1.0          0.003776
INCOME                0.002554
RRACE_3.0             0.001965
polit_1               0.001213
EXPNS_DIF             0.001140
FEWRTRANS_3.0         0.001106
SCHOOL_KIDS_1.0       0.001007
PLNDTRIPS_1.0         0.000682
EIP_4.0               0.000445
MS_5.0                0.000411
LIVQTR_1.0            0.000388
SPNDSRC2_1.0          0.000350
RRACE_4.0             0.000258
HLTHINS1_1.0          0.000237
HLTHINS7_-1.0         0.000205
DOWN_4.0              0.000127
HLTHINS4_-1.0         0.000105
THHLD_NUMADLT_3.0     0.000070
HADCOVID_1.0          0.000069
EST_MSA_41860.0       0.000064
EIP_3.0               0.000056
NOTGET_-1.0           0.000035
HLTHINS5_-1.0         0.000012
EST_MSA_42660.0      -0.000040
DOWN_3.0             -0.000057
LIVQTR_10.0          -0.000114
FEWRTRIPS_1.0        -0.000201
dtype: float64
```

In [653…
```
# get index of importances array in order sorted by mean importance (ascending)
sorted_idx = rf_pi['importances_mean'].argsort()

# Get list of feature names sorted in importance order (ascending)
col_s = pd.Series(final_keep)
col_s[sorted_idx][-20:]
```

Out[653…
```
25      HLTHINS4_-1.0
12           DOWN_4.0
29      HLTHINS7_-1.0
23       HLTHINS1_1.0
40          RRACE_4.0
44       SPNDSRC2_1.0
33         LIVQTR_1.0
35             MS_5.0
15            EIP_4.0
37      PLNDTRIPS_1.0
41    SCHOOL_KIDS_1.0
```

```
20      FEWRTRANS_3.0
19          EXPNS_DIF
55            polit_1
39          RRACE_3.0
31             INCOME
7         CHNGHOW6_1.0
51        TW_START_1.0
13               EEDUC
0                  AGE
dtype: object
```

In [656…

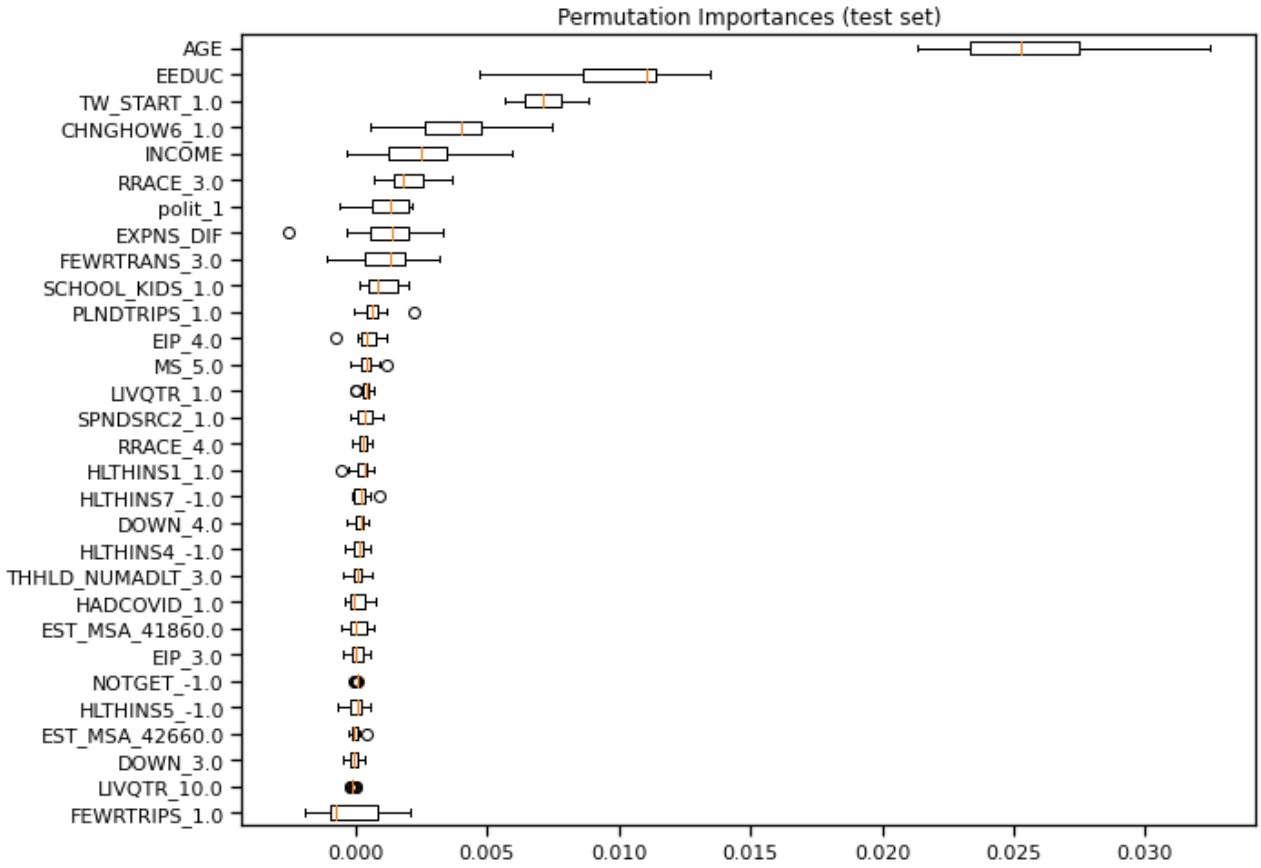
```
# get lists of actual importance scores
rf_pi_sort = rf_pi['importances'][sorted_idx].transpose()
rf_pi_sort[:, -5:]
```

Out[656…

```
array([[ 0.00116134,  0.00744569,  0.007698  ,  0.01136568,  0.03052973],
       [-0.00031845,  0.0040286 ,  0.00622092,  0.01109011,  0.02525055],
       [ 0.00595365,  0.00455458,  0.00881267,  0.01104104,  0.0245974 ],
       [ 0.00247503,  0.00111008,  0.0066753 ,  0.01107882,  0.02811836],
       [ 0.00205217,  0.00376372,  0.00795551,  0.01142736,  0.02678711],
       [ 0.00408356,  0.00245345,  0.00723435,  0.00865559,  0.0238125 ],
       [ 0.00514779,  0.00057317,  0.00880449,  0.0085842 ,  0.02287746],
       [ 0.00325537,  0.00398918,  0.00653236,  0.00837714,  0.02933976],
       [ 0.00106201,  0.00284933,  0.00797304,  0.01346693,  0.02240435],
       [ 0.00131706,  0.00502564,  0.00709891,  0.00906349,  0.02582532],
       [ 0.00333948,  0.00655558,  0.00768958,  0.01233819,  0.03246312],
       [ 0.00270376,  0.00531221,  0.0070287 ,  0.01060562,  0.02207758],
       [ 0.00062461,  0.00193735,  0.00569258,  0.00469163,  0.02530891],
       [ 0.00183976,  0.00431271,  0.00591176,  0.00664472,  0.02502383],
       [ 0.00361342,  0.00273443,  0.00632767,  0.01274753,  0.02136384]])
```

In [847…

```
# Plot top 20 (bottom 20 because sorted from lowest to highest)
# code from https://scikit-learn.org/stable/auto_examples/inspection/plot_permut
fig, ax = plt.subplots(figsize=(10, 8))
ax.boxplot(rf_pi_sort[:, -30:],vert=False, labels=col_s[sorted_idx][-30:])
ax.set_title("Permutation Importances (test set)");
```



Voting on Features

Logistic regression coefficients were interpretable as both odds of hesitancy and odds of optimism.

Using those as a baseline, I compared to ranked features from the Random Forest feature importances and permutation importances.

Out of the 16 categories I identified from LR coefficients, 13 were corroborated by one or the other of the random forest models.

Hesitancy Predictors:

Model Column Name	Description of Question	Odds of Hesitancy
THHLD_NUMKID	Number of individuals under 18 in the household	2.29
CURFOODSUF	Level of household food insufficiency in the past 7 days	1.8
LIVQTR_1.0	Household residence is a mobile home	1.73
FEWRTRANS_3.0	Respondent did not use public transportation such as bus, rail, or ride-share before the pandemic, so transportation did not change	1.51

Optimism Predictors:

Model Column Name	Description of Question	Odds of Optimism

Model Column Name	Description of Question	Odds of Optimism
AGE	Respondent's age in years	6.06
EEDUC	Level of education	2.62
RRACE_3.0	Respondent identified as Asian	2.22
CHNGHOW6_1.0	Members of the household had avoided eating at restaurants in the prior 7 days	2.03
INCOME	Pre-tax income level	1.89
FEWRTRIPS_1.0	Members of the household had taken fewer trips to stores because of the pandemic in the prior 7 days	1.88
TW_START_1.0	At least one adult in the household substituted some or all of their typical in-person work for telework	1.51
ANXIOUS_2.0	Had felt anxious several out of the past 7 days	1.38
HLTHINS3_1.0	Respondent covered by Medicare	1.34

Chi-Squared for categorical features

In [161…
```
import itertools
```

In [ ]:
```
# calculate chi-squared statistic for each categorical predictor
# against the others
```

In [122…
```
def chi2_multicol(df, cols, prob=0.95):
    """Run chi-squared tests for all combinations of columns in cols list
    and returns a dataframe with column pair names, statistic, and p-value.

    This is intended for nominal categorical variables, so pass a list of
    just those columns in the dataframe.

    p-values can be filtered in resulting dataframe to determine which columns
    have NO relationship (null hypotheses) and which columns have SOME
    relationship (alternative hypothesis).

    OHE the categorical columns first to get chi-2 test results on individual
    labels.
    """
    results = []

    cols.sort()

    cat_combos = list(itertools.combinations(cols, 2))

    for combo in cat_combos:

        # get the contingency table using crosstab
        crosstab = pd.crosstab(df[combo[0]], df[combo[1]])

        # use scipy to get chi2 statistic and other info
        stat, p, dof, expected = stats.chi2_contingency(crosstab)

        #https://machinelearningmastery.com/chi-squared-test-for-machine-learnin
        # interpret test-statistic
```

```
        critical = stats.chi2.ppf(prob, dof)
        # append features that are NOT independent of each other to the list
        #if abs(stat) >= critical:
        results.append([combo[0] + '/' + combo[1], stat, p])

    results_df = pd.DataFrame(results, columns=['feature names',
                                                'chi2 stat', 'p-value'])
    return results_df
```

In [121…

```
all_cats = cat_cols + cat_cols_v1 + cat_cols_v2

dep_cats_df = chi2_multicol(df, all_cats)
dep_cats_df
```

Out[121…

	feature names	chi2 stat	p-value
0	ANYWORK/CHNGHOW1	400.885799	1.790222e-85
1	ANYWORK/CHNGHOW10	215.261088	1.961555e-45
2	ANYWORK/CHNGHOW11	233.557296	2.263310e-49
3	ANYWORK/CHNGHOW12	236.024700	6.660081e-50
4	ANYWORK/CHNGHOW2	501.844066	2.674310e-107
…	…	…	…
2011	TW_START/WRKLOSS	870.204536	4.754166e-187
2012	TW_START/inc_binary	461.756280	5.381431e-101
2013	UI_APPLY/WRKLOSS	19391.503073	0.000000e+00
2014	UI_APPLY/inc_binary	938.637695	3.681843e-203
2015	WRKLOSS/inc_binary	137.962492	1.101097e-30

2016 rows × 3 columns

In [121…

```
# which categorical variables are INdependent based on chi2 stats?
dep_cats_df.loc[dep_cats_df['p-value'] > 0.05]
```

Out[121…

	feature names	chi2 stat	p-value
796	EGENDER/EXPCTLOSS	5.061750	0.079589
820	EGENDER/RHISPANIC	2.731537	0.098385
905	ENROLL1/IN_METRO_AREA	2.795804	0.247115
996	ENROLL3/IN_METRO_AREA	3.033528	0.219421

This means almost all of my categorical variables have a dependent relationship between at least one of their respective labels, but I would need to OHE them and do do this again to determine which labels.

In [121…

```
# which individual categorical labels are correlated with each other?
ohe2 = OneHotEncoder(sparse=False)
ohe2.fit(df[all_cats])
df_ohecats = pd.DataFrame(ohe2.transform(df[all_cats]),
```

```
                                columns=ohe2.get_feature_names(df[all_cats].columns))
df_ohecats
```

Out[121…

	ANYWORK_-1.0	ANYWORK_0.0	ANYWORK_1.0	CHNGHOW1_-2.0	CHNGHOW1_0.0	CHNG
0	0.0	1.0	0.0	0.0	0.0	
1	0.0	0.0	1.0	0.0	0.0	
2	0.0	0.0	1.0	1.0	0.0	
3	0.0	0.0	1.0	0.0	1.0	
4	0.0	1.0	0.0	0.0	0.0	
…	…	…	…	…	…	
77820	0.0	0.0	1.0	0.0	1.0	
77821	0.0	1.0	0.0	0.0	1.0	
77822	0.0	1.0	0.0	0.0	0.0	
77823	0.0	0.0	1.0	0.0	0.0	
77824	0.0	1.0	0.0	0.0	1.0	

77825 rows × 249 columns

In [122…

```
# run chi-2 tests on each combo of OHE dummy columns
ohe_dep_cats_df = chi2_multicol(df_ohecats, list(df_ohecats.columns))
ohe_dep_cats_df
```

Out[122…

	feature names	chi2 stat	p-value
0	ANYWORK_-1.0/ANYWORK_0.0	183.685060	7.600918e-42
1	ANYWORK_-1.0/ANYWORK_1.0	324.520393	1.500662e-72
2	ANYWORK_-1.0/CHNGHOW10_-2.0	169.148743	1.135252e-38
3	ANYWORK_-1.0/CHNGHOW10_0.0	59.800162	1.049947e-14
4	ANYWORK_-1.0/CHNGHOW10_1.0	3.227892	7.239382e-02
…	…	…	…
30871	WRKLOSS_0.0/inc_binary_0.0	107.012325	4.424094e-25
30872	WRKLOSS_0.0/inc_binary_1.0	107.012325	4.424094e-25
30873	WRKLOSS_1.0/inc_binary_0.0	94.753452	2.156371e-22
30874	WRKLOSS_1.0/inc_binary_1.0	94.753452	2.156371e-22
30875	inc_binary_0.0/inc_binary_1.0	77818.963638	0.000000e+00

30876 rows × 3 columns

In [122…

```
# which categorical labels are INdependent based on chi2 stats?
ohe_dep_cats_df.loc[ohe_dep_cats_df['p-value'] > 0.05]
```

Out[122…

	feature names	chi2 stat	p-value

	feature names	chi2 stat	p-value
4	ANYWORK_-1.0/CHNGHOW10_1.0	3.227892	0.072394
7	ANYWORK_-1.0/CHNGHOW11_1.0	1.264193	0.260859
12	ANYWORK_-1.0/CHNGHOW1_0.0	1.807380	0.178823
15	ANYWORK_-1.0/CHNGHOW2_0.0	1.893409	0.168818
19	ANYWORK_-1.0/CHNGHOW3_1.0	0.364656	0.545932
...	...	...	...
30823	TW_START_0.0/UI_APPLY_-2.0	1.050736	0.305338
30829	TW_START_0.0/inc_binary_0.0	0.002844	0.957468
30830	TW_START_0.0/inc_binary_1.0	0.002844	0.957468
30840	UI_APPLY_-1.0/UI_APPLY_-2.0	0.004448	0.946827
30844	UI_APPLY_-1.0/WRKLOSS_0.0	1.773020	0.183009

5222 rows × 3 columns

In [ ]: