Assignment 2 Example

Last updated: 9/17/2019, Andrew Therriault

This notebook gives an example of what a completed Week 2 assignment might look like.

Setting up the environment

Setting the working directory

In [1]:
```
import os
os.chdir('c:/working/')
```

Importing numpy and pandas for analysis and setting display options for pandas

In [2]:
```
import numpy as np
import pandas as pd
pd.options.display.max_colwidth = 1000
pd.options.display.max_columns = 1000
pd.options.display.max_rows = 1000
```

Importing seaborn and matplotlib for plotting

Also invoking matplotlib inline cell magic so that all plots will display inline in the notebook (otherwise it only displays one at a time)

In [3]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set(style="whitegrid")
sns.set_color_codes("pastel")
```

Loading Ohio voter file from TargetSmart

This is a 10% sample of the Ohio voter file, with several additional appends from TargetSmart (including their VoterBase demographics and model scores).

In [4]:
```
vf = pd.read_csv('ohio_voterfile_10pct.csv', low_memory=False)
```

Exploring the dataset

Looking at the dimensions (rows, columns)

In [5]:
```
vf.shape
```

Out[5]:
```
(760097, 159)
```

Looking at some sample records

In [6]:
```
vf.sample(5).T.astype(str)
```

Out[6]:

	193937	728929	232322	735524	544988
voterbase_id	OH-000001878359	OH-000002704362	OH-000002123672	OH-10557322	OH-000004984672
tsmart_city	HOLLAND	CINCINNATI	PAINESVILLE	SHADE	CAMDEN
tsmart_state	OH	OH	OH	OH	OH
tsmart_zip	43528	45248	44077	45776	45311
vf_reg_cass_city	HOLLAND	CINCINNATI	PAINESVILLE	SHADE	CAMDEN
vf_reg_cass_state	OH	OH	OH	OH	OH
vf_reg_cass_zip	43528	45248	44077	45776	45311
vf_registration_date	19870316.0	19890717.0	19640917.0	20121106.0	20020515.0
vf_earliest_registration_date	19870316.0	19890717.0	19640917.0	20121106.0	20020515.0
vf_party	Unaffiliated	Unaffiliated	Unaffiliated	Unaffiliated	Unaffiliated
vf_county_code	95	61	85	9	135
vf_county_name	LUCAS	HAMILTON	LAKE	ATHENS	PREBLE
vf_cd	5	1	14	15	8
vf_sd	11	8	18	30	5
vf_hd	46	30	61	94	43
vf_township	SPRINGFIELD UNINCORP	GREEN	PAINESVILLE	LODI	GRATIS
vf_ward	nan	nan	nan	nan	nan
vf_precinct_id	48ASR	31BGP	43AGR	05ABY	68AAN
vf_precinct_name	SPRINGFIELD 12 (48ASR)	GREEN CC (31-BGP)	PAINESVILLE TWP I (43AGR)	LODI TOWNSHIP (05ABY)	GRATIS NORTH (68AAN)
vf_national_precinct_code	OH_LUCAS_48ASR	OH_HAMILTON_31BGP	OH_LAKE_43AGR	OH_ATHENS_05ABY	OH_PREBLE_68AAN
vf_county_council	nan	nan	nan	nan	nan
vf_city_council	nan	nan	nan	nan	nan
vf_municipal_district	nan	nan	nan	nan	nan
vf_school_district	SPRINGFIELD LOCAL SD (LUCAS)	OAK HILLS LOCAL SD (HAMILTON)	RIVERSIDE LOCAL SD (LAKE)	ALEXANDER LOCAL SD (ATHENS)	PREBLE SHAWNEE LOCAL SD (PREBLE)
vf_judicial_district	nan	nan	nan	nan	nan
reg_census_id	390950000000000.0	390610000000000.0	390852000000000.0	390100000000000.0	391355000000000.0
reg_dma	547.0	515.0	510.0	535.0	542.0
reg_dma_name	Toledo OH	Cincinnati OH	Cleveland-Akron (Canton) OH	Columbus OH	Dayton OH
reg_place	nan	3921742.0	nan	nan	nan
reg_place_name	nan	Dent	nan	nan	nan
tsmart_county_code	95.0	61.0	85.0	9.0	135.0
tsmart_county_name	LUCAS	HAMILTON	LAKE	ATHENS	PREBLE
tsmart_cd	5.0	1.0	14.0	15.0	8.0
tsmart_sd	11.0	8.0	18.0	30.0	5.0
tsmart_hd	46.0	30.0	61.0	94.0	43.0
tsmart_township	SPRINGFIELD UNINCORP	GREEN	PAINESVILLE	LODI	GRATIS
tsmart_ward	nan	nan	nan	nan	nan
tsmart_precinct_id	48ASR	31BGP	43AGR	05ABY	68AAN
tsmart_precinct_name	SPRINGFIELD 12 (48ASR)	GREEN CC (31-BGP)	PAINESVILLE TWP I (43AGR)	LODI TOWNSHIP (05ABY)	GRATIS NORTH (68AAN)
tsmart_county_council	nan	nan	nan	nan	nan
tsmart_city_council	nan	nan	nan	nan	nan
tsmart_municipal_district	nan	nan	nan	nan	nan
tsmart_school_district	SPRINGFIELD LOCAL SD (LUCAS)	OAK HILLS LOCAL SD (HAMILTON)	RIVERSIDE LOCAL SD (LAKE)	ALEXANDER LOCAL SD (ATHENS)	PREBLE SHAWNEE LOCAL SD (PREBLE)
tsmart_judicial_district	nan	nan	nan	nan	nan
tsmart_census_id	390950000000000.0	390610000000000.0	390852000000000.0	390100000000000.0	391355000000000.0
tsmart_dma	547.0	515.0	510.0	535.0	542.0
tsmart_dma_name	Toledo OH	Cincinnati OH	Cleveland-Akron (Canton) OH	Columbus OH	Dayton OH
tsmart_place	nan	3921742.0	nan	nan	nan
tsmart_place_name	nan	Dent	nan	nan	nan
vf_g2018	Y	Y	Y	nan	Y
vf_g2017	Y	Y	Y	nan	Y

	193937	728929	232322	735524	544988
vf_g2016	Y	Y	Y	Y	Y
vf_g2015	Y	Y	Y	Y	Y
vf_g2014	Y	Y	Y	nan	nan
vf_g2013	nan	Y	Y	nan	nan
vf_g2012	Y	Y	Y	Y	Y
vf_g2011	Y	Y	Y	nan	Y
vf_g2010	Y	Y	Y	nan	Y
vf_g2009	Y	Y	Y	nan	Y
vf_g2008	Y	Y	Y	nan	Y
vf_g2007	Y	Y	nan	nan	nan
vf_g2006	Y	Y	Y	nan	nan
vf_g2005	Y	Y	Y	nan	nan
vf_g2004	Y	Y	Y	nan	Y
vf_g2003	Y	Y	Y	nan	nan
vf_g2002	Y	Y	Y	nan	nan
vf_g2001	Y	Y	Y	nan	nan
vf_g2000	Y	Y	Y	nan	nan
vf_p2019	nan	nan	nan	nan	nan
vf_p2019_party	nan	nan	nan	nan	nan
vf_p2018	nan	Y	Y	nan	nan
vf_p2018_party	nan	R	D	nan	nan
vf_p2017	nan	Y	nan	nan	Y
vf_p2017_party	nan	nan	nan	nan	nan
vf_p2016	Y	Y	Y	Y	Y
vf_p2016_party	R	R	D	D	R
vf_p2015	nan	nan	nan	nan	nan
vf_p2015_party	nan	nan	nan	nan	nan
vf_p2014	nan	Y	Y	nan	nan
vf_p2014_party	nan	R	D	nan	nan
vf_p2013	nan	nan	nan	nan	nan
vf_p2013_party	nan	nan	nan	nan	nan
vf_p2012	Y	Y	Y	nan	nan
vf_p2012_party	R	R	D	nan	nan
vf_p2011	nan	nan	Y	nan	nan
vf_p2011_party	nan	nan	nan	nan	nan
vf_p2010	Y	Y	Y	nan	nan
vf_p2010_party	R	R	D	nan	nan
vf_p2009	nan	nan	nan	nan	nan
vf_p2009_party	nan	nan	nan	nan	nan
vf_p2008	nan	Y	Y	nan	Y
vf_p2008_party	nan	R	D	nan	D
vf_p2007	nan	nan	nan	nan	nan
vf_p2007_party	nan	nan	nan	nan	nan
vf_p2006	Y	nan	Y	nan	Y
vf_p2006_party	nan	nan	D	nan	nan
vf_p2005	nan	nan	Y	nan	nan
vf_p2005_party	nan	nan	nan	nan	nan
vf_p2004	nan	Y	Y	nan	nan
vf_p2004_party	nan	R	D	nan	nan
vf_p2003	nan	nan	Y	nan	nan
vf_p2003_party	nan	nan	nan	nan	nan
vf_p2002	nan	Y	nan	nan	nan
vf_p2002_party	nan	R	nan	nan	nan
vf_p2001	nan	nan	nan	nan	nan
vf_p2001_party	nan	nan	nan	nan	nan
vf_p2000	Y	Y	Y	nan	nan

	193937	728929	232322	735524	544988
vf_p2000_party	R	R	D	nan	nan
vf_m2019	nan	nan	nan	nan	nan
vf_m2018	nan	nan	nan	nan	nan
vf_m2017	nan	nan	nan	nan	nan
vf_m2016	nan	nan	nan	nan	nan
vf_m2015	nan	nan	nan	nan	nan
vf_m2014	nan	nan	nan	nan	nan
vf_m2013	nan	nan	nan	nan	nan
vf_m2012	nan	nan	nan	nan	nan
vf_m2011	nan	nan	nan	nan	nan
vf_m2010	nan	nan	nan	nan	nan
vf_m2009	nan	nan	nan	nan	nan
vf_m2008	nan	nan	nan	nan	nan
vf_m2007	nan	nan	nan	nan	nan
vf_m2006	nan	nan	nan	nan	nan
vf_m2005	nan	nan	nan	nan	nan
vf_m2004	nan	nan	nan	nan	nan
vf_m2003	nan	nan	nan	nan	nan
vf_m2002	nan	nan	nan	nan	nan
vf_m2001	nan	nan	nan	nan	nan
vf_m2000	nan	nan	nan	nan	nan
vf_pp2020	nan	nan	nan	nan	nan
vf_pp2020_party	nan	nan	nan	nan	nan
vf_pp2016	nan	nan	nan	nan	nan
vf_pp2016_party	nan	nan	nan	nan	nan
vf_pp2012	nan	nan	nan	nan	nan
vf_pp2012_party	nan	nan	nan	nan	nan
vf_pp2008	nan	Y	Y	nan	Y
vf_pp2008_party	nan	nan	nan	nan	nan
vf_pp2004	nan	nan	nan	nan	nan
vf_pp2004_party	nan	nan	nan	nan	nan
vf_pp2000	nan	nan	nan	nan	nan
vf_pp2000_party	nan	nan	nan	nan	nan
tsmart_partisan_score	1.2	1.2	99.2	99.3	9.7
tsmart_presidential_general_turnout_score	98.6	98.4	97.5	85.7	95.9
tsmart_midterm_general_turnout_score	90.7	96.6	95.7	31.3	81.1
tsmart_midterm_general_enthusiasm_score	85.4	86.6	90.6	22.9	49.1
tsmart_offyear_general_turnout_score	84.0	89.8	86.3	11.1	62.7
tsmart_presidential_primary_turnout_score	78.5	92.0	90.4	38.7	56.0
tsmart_non_presidential_primary_turnout_score	37.0	76.3	73.1	6.3	27.4
voterbase_age	66.0	62.0	82.0	30.0	56.0
voterbase_gender	Male	Female	Female	Male	Male
voterbase_race	Caucasian	Caucasian	Caucasian	Caucasian	Caucasian
voterbase_marital_status	Married	Married	Married	Unmarried	Married
vf_voter_status	Active	Active	Active	Active	Active
voterbase_deceased_flag	nan	nan	nan	nan	nan
deceased_flag_date_of_death	nan	nan	nan	nan	nan
voterbase_mailable_flag	Yes	Yes	Yes	Yes	Yes
vf_missing_occupancy_flag	nan	nan	nan	nan	nan
vf_absentee_status	nan	nan	nan	nan	nan
vf_early_voter_status	nan	nan	nan	nan	nan
vf_pav	nan	nan	nan	nan	nan

Looking at the distributions of some key demographic and political variables

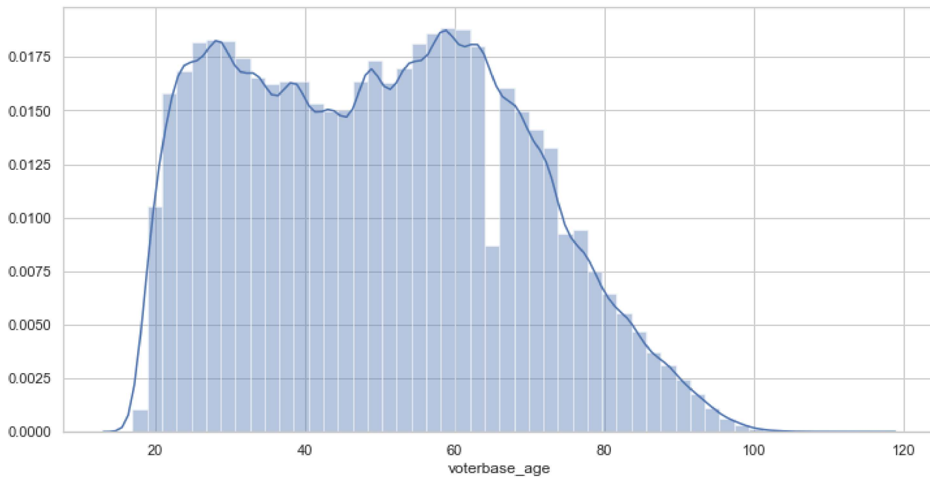
In [7]:
```
vf.voterbase_age.describe()
```

Out[7]:
```
count    760085.000000
mean         49.902715
std          18.508834
min          17.000000
25%          34.000000
50%          50.000000
75%          64.000000
max         115.000000
Name: voterbase_age, dtype: float64
```

Note that for the seaborn distribution plot, we need to limit to non-null values.

In [8]:
```
f, ax = plt.subplots(1, figsize=(12,6))
sns.distplot(vf.voterbase_age[vf.voterbase_age.notnull()])
```

Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bb84e99ef0>
```



Filling in null values with "Missing" for calculating proportions across the full population

In [9]:
```
for i in ['gender','race','marital_status','deceased_flag']:
    print('\nProportions by {}'.format(i))
    print(vf['voterbase_{}'.format(i)].fillna('Missing').value_counts(normalize=True))
```

```
Proportions by gender
Female     0.486867
Male       0.453746
Unknown    0.059387
Name: voterbase_gender, dtype: float64

Proportions by race
Caucasian           0.876057
African-American    0.092105
Uncoded             0.022166
Hispanic            0.006036
Asian               0.003513
Native American     0.000124
Name: voterbase_race, dtype: float64

Proportions by marital_status
Unmarried    0.503740
Married      0.442940
Unknown      0.053321
Name: voterbase_marital_status, dtype: float64

Proportions by deceased_flag
Missing     0.999471
Deceased    0.000529
Name: voterbase_deceased_flag, dtype: float64
```

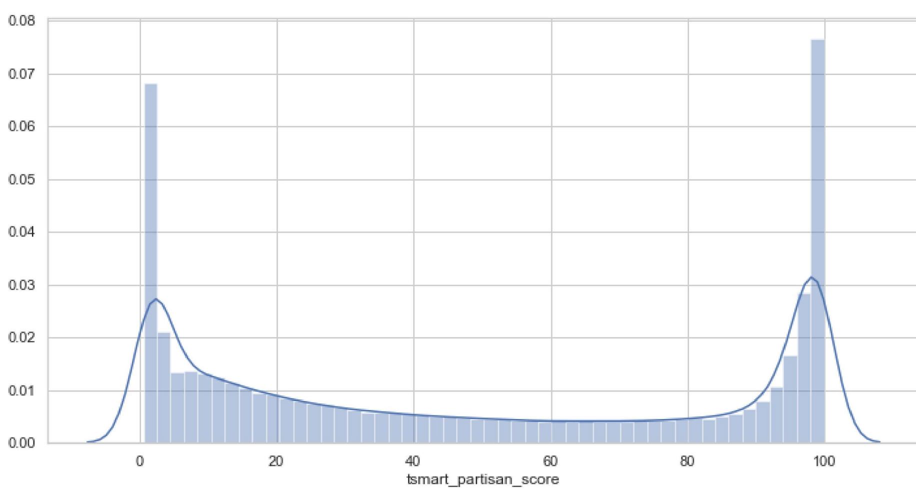
In [10]:
```
for i in ['voter_status','absentee_status','g2018', 'g2016', 'p2018_party']:
    print('\nProportions by {}'.format(i))
    print(vf['vf_{}'.format(i)].fillna('Missing').value_counts(normalize=True))
```

```
Proportions by voter_status
Active      0.899237
Inactive    0.100763
Name: vf_voter_status, dtype: float64

Proportions by absentee_status
Missing    0.864488
Yes        0.135512
Name: vf_absentee_status, dtype: float64

Proportions by g2018
Y          0.582862
Missing    0.403435
Z          0.008625
R          0.002542
B          0.001558
F          0.000950
S          0.000028
Name: vf_g2018, dtype: float64

Proportions by g2016
Y          0.669586
Missing    0.294211
Z          0.026668
R          0.004481
B          0.002933
F          0.002094
S          0.000028
Name: vf_g2016, dtype: float64

Proportions by p2018_party
Missing    0.794933
R          0.111803
D          0.092745
G          0.000520
Name: vf_p2018_party, dtype: float64
```

In [11]:
```
print(vf.tsmart_partisan_score.describe())
f, ax = plt.subplots(1, figsize=(12,6))
sns.distplot(vf.tsmart_partisan_score)
```

```
count    760097.000000
mean         48.399587
std          38.356573
min           0.500000
25%           9.900000
50%          41.400000
75%          93.300000
max          99.900000
Name: tsmart_partisan_score, dtype: float64
```

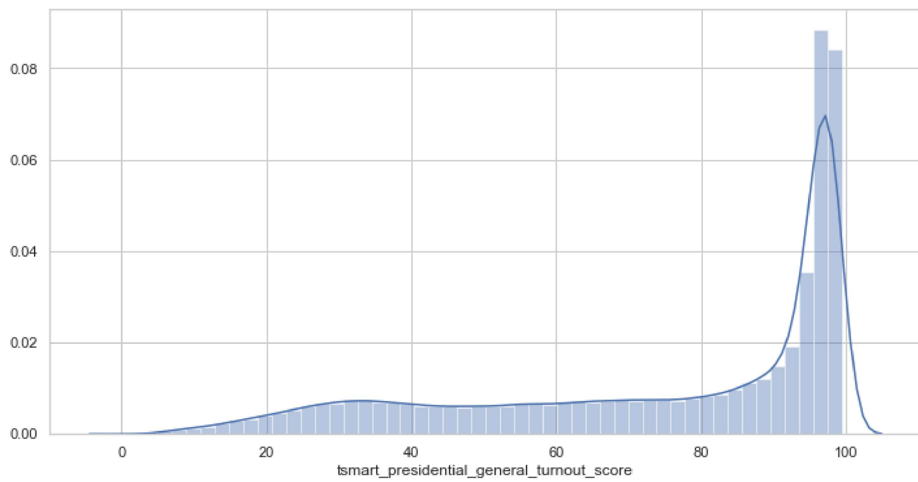
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x1bb80935128>



In [12]:
```
print(vf.tsmart_presidential_general_turnout_score.describe())
f, ax = plt.subplots(1, figsize=(12,6))
sns.distplot(vf.tsmart_presidential_general_turnout_score)
```

```
count    760097.000000
mean         74.766173
std          26.097281
min           1.100000
25%          55.000000
50%          87.600000
75%          96.600000
max          99.400000
Name: tsmart_presidential_general_turnout_score, dtype: float64
```

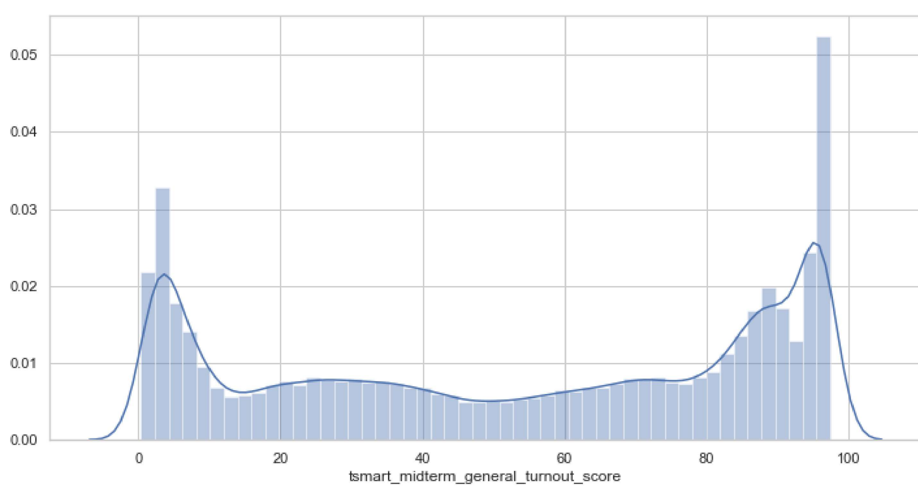
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x1bb808c6d68>



In [13]:
```
print(vf.tsmart_midterm_general_turnout_score.describe())
f, ax = plt.subplots(1, figsize=(12,6))
sns.distplot(vf.tsmart_midterm_general_turnout_score)
```

```
count    760097.000000
mean         53.346607
std          34.278661
min           0.400000
25%          20.100000
50%          58.400000
75%          87.400000
max          97.400000
Name: tsmart_midterm_general_turnout_score, dtype: float64
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x1bb80ab0400>



Question 1: How do the 2016 and 2018 electorates vary in terms of demographics from the overall registered voter populations?

Flagging electorates in each year as well as those eligible (based on earliest registration date)

```
In [14]: vf['voted16'] = (vf.vf_g2016.notnull())
         vf['eligible16'] = (vf.vf_earliest_registration_date <= 20161108)
         vf['voted18'] = (vf.vf_g2018.notnull())
         vf['eligible18'] = (vf.vf_earliest_registration_date <= 20181106)
```

Calculating distributions for each demographic category

```
In [15]: for i in ['gender','race','marital_status']:
             print('\nBy {}:'.format(i))
             x = pd.DataFrame()
             for j in ['voted16','eligible16','voted18','eligible18']:
                 x[j] = vf.loc[vf[j],'voterbase_{}'.format(i)].value_counts(normalize=True).sort_index()
             print(x)
```

```
By gender:
          voted16  eligible16   voted18  eligible18
Female   0.512006    0.495024  0.503062    0.486324
Male     0.451911    0.457755  0.460482    0.455502
Unknown  0.036082    0.047222  0.036456    0.058174

By race:
                   voted16  eligible16   voted18  eligible18
African-American  0.077386    0.090700  0.073003    0.090368
Asian             0.003016    0.003138  0.003231    0.003490
Caucasian         0.897409    0.880237  0.903080    0.877808
Hispanic          0.004153    0.005834  0.003189    0.006152
Native American   0.000110    0.000124  0.000106    0.000124
Uncoded           0.017927    0.019966  0.017391    0.022058

By marital_status:
            voted16  eligible16   voted18  eligible18
Married    0.525248    0.460207  0.546172    0.445340
Unknown    0.040012    0.045799  0.035949    0.044108
Unmarried  0.434740    0.493994  0.417879    0.510551
```

Calculating distributions by age

```
In [16]: for j in ['voted16','eligible16','voted18','eligible18']:
             print('Average age (current) for {} = {}'.format(j, np.round(vf[vf[j]].voterbase_age.mean(),1)))
```

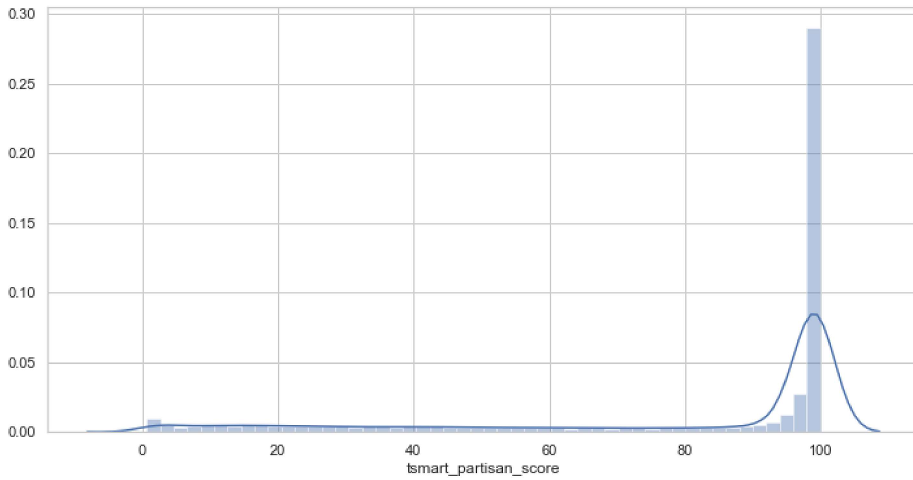
```
Average age (current) for voted16 = 53.9
Average age (current) for eligible16 = 51.1
Average age (current) for voted18 = 54.7
Average age (current) for eligible18 = 49.6
```

Question 2: Compare partisanship scores across Dem primary voters in 2018, Rep primary voters, and those who didn't vote in a primary

In [17]:
```
dem_voter_scores = vf[vf.vf_p2018_party=='D'].tsmart_partisan_score
print(dem_voter_scores.describe())
f, ax = plt.subplots(1, figsize=(12,6))
sns.distplot(dem_voter_scores)
```

```
count    70495.000000
mean        79.656075
std         31.400597
min          0.700000
25%         64.700000
50%         98.800000
75%         99.300000
max         99.900000
Name: tsmart_partisan_score, dtype: float64
```

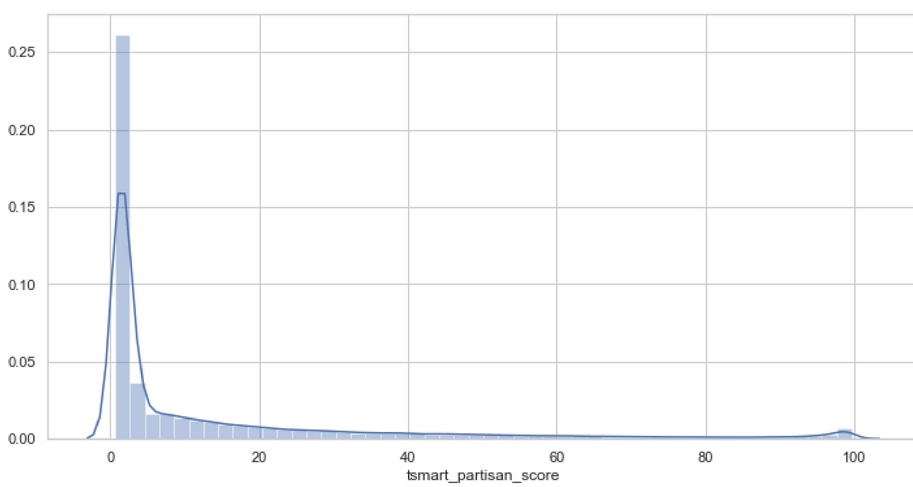
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x1bb80f87d68>



In [18]:
```
rep_voter_scores = vf[vf.vf_p2018_party=='R'].tsmart_partisan_score
print(rep_voter_scores.describe())
f, ax = plt.subplots(1, figsize=(12,6))
sns.distplot(rep_voter_scores)
```

```
count    84981.000000
mean        14.150463
std         22.809783
min          0.600000
25%          1.300000
50%          2.300000
75%         16.600000
max         99.700000
Name: tsmart_partisan_score, dtype: float64
```

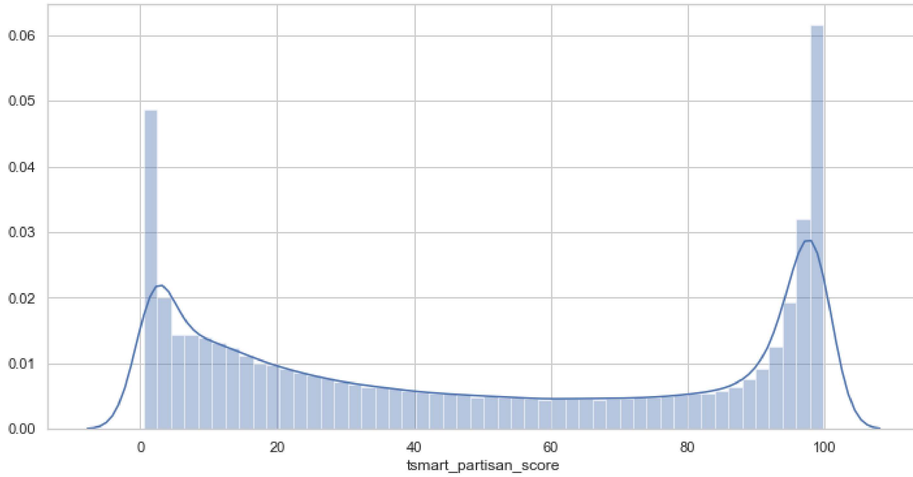
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x1bb80d4d710>



In [19]:
```
non_voter_scores = vf[vf.vf_p2018_party.isnull()].tsmart_partisan_score
print(non_voter_scores.describe())
f, ax = plt.subplots(1, figsize=(12,6))
sns.distplot(non_voter_scores)
```

```
count    604226.000000
mean         49.563015
std          37.163721
min           0.500000
25%          12.700000
50%          44.400000
75%          91.900000
max          99.900000
Name: tsmart_partisan_score, dtype: float64
```

Out[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bb80f63b00>
```



Question 3: How are partisanship scores correlated with demographics?

Splitting ages into groups

In [20]:
```
vf['voterbase_age_group'] = pd.qcut(vf.voterbase_age,5).astype(str)
```

Describing scores by group

In [21]:
```
for i in ['gender','race','marital_status','age_group']:
    print('\nBy {}:'.format(i))
    x = pd.DataFrame()
    for j in sorted(vf['voterbase_{}'.format(i)].unique()):
        x[j] = vf.loc[vf['voterbase_{}'.format(i)]==j,'tsmart_partisan_score'].describe()
    print(x)
```

```
By gender:
              Female           Male       Unknown
count  370066.000000  344891.000000  45140.000000
mean       52.479572      41.511753     67.577448
std        38.477440      37.359184     33.247574
min         0.600000       0.500000      0.600000
25%        12.500000       7.100000     38.800000
50%        51.900000      27.300000     82.600000
75%        95.700000      83.900000     97.100000
max        99.900000      99.900000     99.900000

By race:
       African-American         Asian      Caucasian     Hispanic  \
count      70009.000000  2670.000000  665888.000000  4588.000000
mean          94.498537    60.929775      42.857654    81.939037
std           13.904150    32.205449      36.807006    24.306787
min            1.100000     0.900000       0.500000     0.800000
25%           95.900000    32.500000       7.800000    79.800000
50%           97.900000    68.500000      31.900000    92.100000
75%           99.200000    91.300000      82.100000    96.900000
max           99.900000    99.800000      99.900000    99.800000

       Native American       Uncoded
count        94.000000  16848.000000
mean         66.572340     64.658060
std          33.692932     32.457174
min           1.100000      0.700000
25%          44.400000     38.100000
50%          80.500000     75.400000
75%          95.875000     94.300000
max          99.600000     99.900000

By marital_status:
             Married       Unknown      Unmarried
count  336677.000000  40529.000000  382891.000000
mean       35.453749     55.610923      59.019576
std        36.875594     36.652535      36.291559
min         0.500000      0.700000       0.600000
25%         3.400000     18.900000      22.800000
50%        18.500000     58.700000      66.900000
75%        66.800000     94.800000      96.000000
max        99.900000     99.900000      99.900000

By age_group:
       (16.999, 31.0]   (31.0, 43.0]   (43.0, 56.0]   (56.0, 67.0]  \
count   160721.000000  144541.000000  160339.000000  147479.000000
mean        57.525450      52.050759      44.032095      44.146287
std         34.984921      36.694011      38.118733      39.520413
min          0.600000       0.500000       0.600000       0.600000
25%         22.400000      15.500000       7.800000       5.200000
50%         63.600000      50.100000      31.200000      30.200000
75%         93.000000      92.900000      90.300000      94.000000
max         99.900000      99.900000      99.800000      99.800000

       (67.0, 115.0]        nan
count  147005.000000  12.000000
mean       43.862363  55.458333
std        40.367439  36.712135
min         0.600000   2.200000
25%         3.100000  27.475000
50%        29.000000  64.850000
75%        95.700000  89.350000
max        99.800000  98.900000
```

Question 4: Graph new / updated registrations by month over the past 8 years

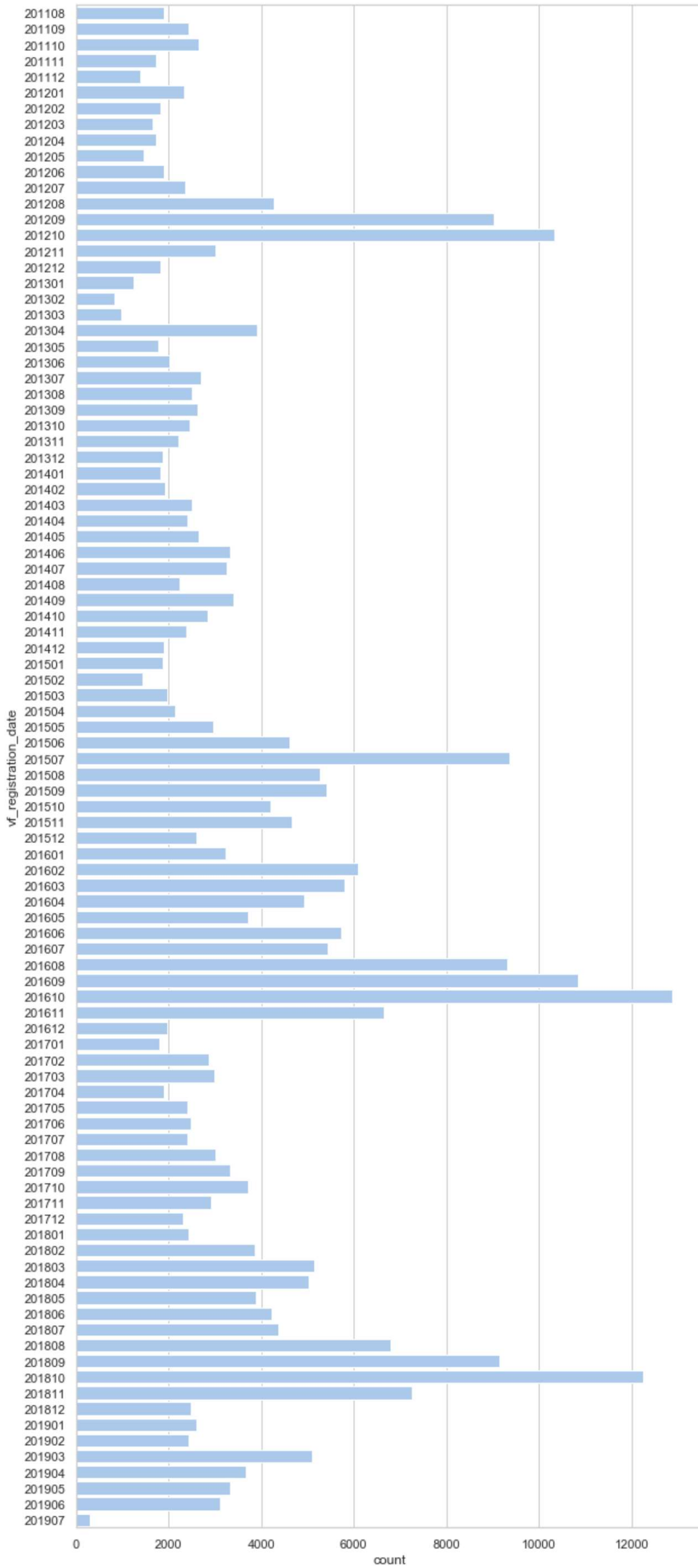
Extracting month and year from registration dates and limiting to last 8 years

In [22]:
```
reg_dates = vf.vf_registration_date.astype(str).map(lambda x: x[:6])
reg_dates = reg_dates[(reg_dates.astype(float) > 201107)]
reg_by_month = reg_dates.value_counts().sort_index()
```

Plotting registrations by month

```
In [23]: f, ax = plt.subplots(figsize=(10, 25))
         sns.countplot(y=reg_dates, color='b')
```

```
Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x1bb93546f98>
```



Question 5: How does the population of inactive registrants vary across counties?

Calculating percentage of inactives by county

In [24]:
```
inactive_pct_by_county = 100*(vf.vf_voter_status == 'Inactive').astype(int).groupby(
                                vf.vf_county_name).mean().sort_values(ascending=False)
```

Plotting across counties

In [25]:
```
f, ax = plt.subplots(figsize=(10, 25))
sns.barplot(x=inactive_pct_by_county, y=inactive_pct_by_county.index, color='b')
```

Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x1bb9353ca58>