Output1.2

June 2, 2021

1 Data Set

Survey data aimed capture how average citizen spend their entire day doing various chores. Data has responses from 64K respondees between year 2005 to 2012.

1.1 Objective of the Study

1. Data Exploration
2. Data Cleaning by addressing outliers in the dataset
3. Obtain information regarding how employeed and non employeed people speend their day using statistical modeling
   a. Correlation Analysis between continuous and discrete variables
   b. Use Anova and Chisquare Tets fro obtaining dependancies between Categorical and Continuous variables
   c. Interprete Results
4. Developed a machine learning model to predict employment status of indiuals based on their time spent patterns
5. Create presentation using Tableau and Powerpoint to summarize the findings

[3]:
```
# Import Libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as mlt
import seaborn as snr
import warnings
warnings.filterwarnings('ignore')
```

[4]:
```
df=pd.read_excel('./Training Dataset.xlsx')
```

[14]:
```
df['Year'].describe()
```

[14]:
```
count    64006.000000
mean      2008.500109
std          2.291258
```

```
min        2005.000000
25%        2007.000000
50%        2009.000000
75%        2010.750000
max        2012.000000
Name: Year, dtype: float64
```

[4]:
```python
df.head()
```

[4]:
```
   Id Education Level  Age Age Range   Employment Status  Gender  Children  \
0   1     High School   51     50-59          Unemployed  Female         0
1   2        Bachelor   42     40-49            Employed  Female         2
2   3          Master   47     40-49            Employed    Male         0
3   4    Some College   21     20-29            Employed  Female         0
4   5     High School   49     40-49  Not in labor force  Female         0

   Weekly Earnings  Year  Weekly Hours Worked  ...  Playing with Children  \
0                0  2005                    0  ...                      0
1             1480  2005                   40  ...                     20
2              904  2005                   40  ...                      0
3              320  2005                   40  ...                      0
4                0  2005                    0  ...                      0

   Job Searching  Shopping  Eating and Drinking  Socializing & Relaxing  \
0              0         0                   40                     180
1              0       120                   40                      15
2              0        15                   85                     214
3              0       105                   30                     240
4              0         0                   35                     600

   Television  Golfing  Running  Volunteering      Total
0         120        0        0             0  24.000000
1          15        0        0             0  21.583333
2         199        0        0             0  17.733333
3         240        0        0             0  26.833333
4          40        0        0             0  23.750000

[5 rows x 25 columns]
```

[5]:
```python
df.shape
```

[5]:
```
(64006, 25)
```

[6]:
```python
df.isnull().sum()/len(df)
```

[6]:
```
Id                    0.0
Education Level       0.0
Age                   0.0
Age Range             0.0
Employment Status     0.0
```

```
Gender                    0.0
Children                  0.0
Weekly Earnings           0.0
Year                      0.0
Weekly Hours Worked       0.0
Sleeping                  0.0
Grooming                  0.0
Housework                 0.0
Food & Drink Prep         0.0
Caring for Children       0.0
Playing with Children     0.0
Job Searching             0.0
Shopping                  0.0
Eating and Drinking       0.0
Socializing & Relaxing    0.0
Television                0.0
Golfing                   0.0
Running                   0.0
Volunteering              0.0
Total                     0.0
dtype: float64
```

[7]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 64006 entries, 0 to 64005
Data columns (total 25 columns):
Id                        64006 non-null int64
Education Level           64006 non-null object
Age                       64006 non-null int64
Age Range                 64006 non-null object
Employment Status         64006 non-null object
Gender                    64006 non-null object
Children                  64006 non-null int64
Weekly Earnings           64006 non-null int64
Year                      64006 non-null int64
Weekly Hours Worked       64006 non-null int64
Sleeping                  64006 non-null int64
Grooming                  64006 non-null int64
Housework                 64006 non-null int64
Food & Drink Prep         64006 non-null int64
Caring for Children       64006 non-null int64
Playing with Children     64006 non-null int64
Job Searching             64006 non-null int64
Shopping                  64006 non-null int64
Eating and Drinking       64006 non-null int64
Socializing & Relaxing    64006 non-null int64
Television                64006 non-null int64
```

```
Golfing                    64006 non-null int64
Running                    64006 non-null int64
Volunteering               64006 non-null int64
Total                      64006 non-null float64
dtypes: float64(1), int64(20), object(4)
memory usage: 12.2+ MB
```

[8]:
```
df.describe()
```

[8]:
```
                 Id           Age      Children  Weekly Earnings  \
count  64006.000000  64006.000000  64006.000000     64006.000000
mean   32003.500000     46.260569      0.891291       485.697872
std    18477.085002     17.396500      1.146851       639.891303
min        1.000000     15.000000      0.000000         0.000000
25%    16002.250000     33.000000      0.000000         0.000000
50%    32003.500000     45.000000      0.000000       240.000000
75%    48004.750000     59.000000      2.000000       769.000000
max    64006.000000     85.000000     12.000000      2885.000000

               Year  Weekly Hours Worked      Sleeping      Grooming  \
count  64006.000000         64006.000000  64006.000000  64006.000000
mean    2008.500109            24.508796    522.240368     40.591116
std        2.291258            22.274917    135.669820     36.713372
min     2005.000000             0.000000      0.000000      0.000000
25%     2007.000000             0.000000    445.000000     10.000000
50%     2009.000000            30.000000    510.000000     30.000000
75%     2010.750000            40.000000    600.000000     60.000000
max     2012.000000           160.000000   1423.000000   1043.000000

          Housework  Food & Drink Prep  ...  Playing with Children  \
count  64006.000000       64006.000000  ...           64006.000000
mean      41.246618          34.287879  ...               8.498172
std       82.483654          53.508507  ...              39.001215
min        0.000000           0.000000  ...               0.000000
25%        0.000000           0.000000  ...               0.000000
50%        0.000000          10.000000  ...               0.000000
75%       55.000000          50.000000  ...               0.000000
max     1405.000000         995.000000  ...             840.000000

       Job Searching      Shopping  Eating and Drinking  \
count   64006.000000  64006.000000         64006.000000
mean        1.700606     24.668234            68.652189
std        20.706929     49.144949            52.639850
min         0.000000      0.000000             0.000000
25%         0.000000      0.000000            30.000000
50%         0.000000      0.000000            60.000000
75%         0.000000     30.000000            90.000000
max       983.000000    879.000000           895.000000
```

```
       Socializing & Relaxing    Television       Golfing       Running  \
count                64006.000000  64006.000000  64006.000000  64006.000000
mean                   288.137925    165.160735      1.293191      0.686201
std                    206.163299    168.431664     18.539409      7.421383
min                      0.000000      0.000000      0.000000      0.000000
25%                    125.000000     30.000000      0.000000      0.000000
50%                    250.000000    120.000000      0.000000      0.000000
75%                    414.000000    240.000000      0.000000      0.000000
max                   1434.000000   1380.000000    600.000000    505.000000

       Volunteering         Total
count  64006.000000  64006.000000
mean       9.831953     20.615301
std       49.762815      6.194366
min        0.000000      0.000000
25%        0.000000     15.833333
50%        0.000000     20.500000
75%        0.000000     24.833333
max     1127.000000     46.666667

[8 rows x 21 columns]
```

[3]:
```python
df=df.drop(['Id'], axis=1)
```

[4]:
```python
df=df.drop(['Total'], axis=1)
```

[5]:
```python
# lets check for unique values

for col in df:
    print(col, df[col].unique()[:20],'\n')
```

```
Education Level ['High School' 'Bachelor' 'Master' 'Some College' '11th grade'
 'Associate Degree' '9th grade' '10th grade' 'Prof. Degree' '12th grade'
 'Doctoral Degree']

Age [51 42 47 21 49 44 46 24 30 26 32 43 22 80 62 39 38 57 40 61]

Age Range ['50-59' '40-49' '20-29' '30-39' '80+' '60-69' '0-19' '70-79']

Employment Status ['Unemployed' 'Employed' 'Not in labor force']

Gender ['Female' 'Male']

Children [ 0  2  1  4  3  5  6  8  7 11 10  9 12]

Weekly Earnings [   0 1480  904  320  700  442  378 1000  100  396  788  380
1385  485
  769 1202 1269  962  316  500]
```

```
Year [2005 2006 2007 2008 2009 2010 2011 2012]

Weekly Hours Worked [  0  40  45  60  10  71  30  55  50  20  16  35  15  52  61
47   5  38
 100  29]

Sleeping [ 825  500  480  705  470  750  445  435  510  539 1260  720  430  870
  277  490  390  457  570  780]

Grooming [ 80  10  70  65  60  20   0  30  57 120  90  40  67  25  15  75  50
45
  85   5]

Housework [150   1  90 205   0 270  45 195  30 120   5 210 105 220 240  60  10
40
 175  15]

Food & Drink Prep [ 45  60   0 135  15  70  30 255 160   3 120   1   5  22  10
25 108  20
  80  35]

Caring for Children [  0 365 120 147  33  70 205   1  15   5  99 240  35  64 110
59 140 137
 170  51]

Playing with Children [  0  20 225  50  60 135 120  65  40 375 240 150  30  70
332 350 180  86
  45 190]

Job Searching [  0  90  60  15 225  50 220 210  30 180 343 300 570 150  20 120
490 105
 240 134]

Shopping [  0 120  15 105  20  80  30  25  35  90 165   5  45  60  10   1 110
115
 390 128]

Eating and Drinking [ 40  85  30  35  15  60   0  90 140  45   5  99 125  70  20
32  69 135
 126 150]

Socializing & Relaxing [180  15 214 240 600 596 165   0 225 330 313 450 120 300
270 440 706 480
 470 265]

Television [120  15 199 240  40  84 150   0  90 210 180  75 290 474 200 410 265
208
```

```
 380 360]

Golfing [  0 140 240 255 270 490 252  60  30 345 260 235 330 150 137 120 315 360
  75 210]

Running [  0  30  28  25  80  60  40  35  90  20  45  15 180  50   5   7 170 120
 105 125]

Volunteering [  0  15 185  88  10 297 135  53 145  81  60 190  30   5 335 891
150  90
 164  20]
```

[6]:
```python
#Lets convert year into objct
df['Year']=df['Year'].astype('O')
```

[7]:
```python
# lets check for employed individuals

employ=df[df['Employment Status']=='Employed']

#We see that the minimum value for weekly hours worked and weekly earnings is 0.
↪ Something is wrong here.
employ.describe()
```

[7]:
```
                Age      Children  Weekly Earnings  Weekly Hours Worked  \
count  41098.000000  41098.000000     41098.000000         41098.000000
mean      42.385712      0.978320       756.425568            38.169984
std       13.238091      1.131968       657.958465            15.851958
min       15.000000      0.000000         0.000000             0.000000
25%       33.000000      0.000000       288.000000            35.000000
50%       42.000000      1.000000       610.000000            40.000000
75%       52.000000      2.000000      1050.000000            45.000000
max       85.000000     10.000000      2885.000000           160.000000

           Sleeping      Grooming     Housework  Food & Drink Prep  \
count  41098.000000  41098.000000  41098.000000       41098.000000
mean     508.438415     41.668986     36.042168          29.087547
std      129.681133     35.025107     76.393757          47.393238
min        0.000000      0.000000      0.000000           0.000000
25%      430.000000     15.000000      0.000000           0.000000
50%      500.000000     35.000000      0.000000           6.500000
75%      577.000000     60.000000     35.000000          45.000000
max     1405.000000   1043.000000   1030.000000         995.000000

       Caring for Children  Playing with Children  Job Searching  \
count         41098.000000           41098.000000   41098.000000
mean             30.725923               8.794175       0.506983
std              73.363468              39.167168       9.858554
```

```
min            0.000000                  0.000000      0.000000
25%            0.000000                  0.000000      0.000000
50%            0.000000                  0.000000      0.000000
75%           20.000000                  0.000000      0.000000
max          985.000000                760.000000    630.000000

           Shopping  Eating and Drinking  Socializing & Relaxing  \
count  41098.000000         41098.000000            41098.000000
mean      24.724244            67.754076              235.931335
std       48.938202            52.503347              177.516482
min        0.000000             0.000000                0.000000
25%        0.000000            30.000000              105.000000
50%        0.000000            60.000000              200.000000
75%       30.000000            90.000000              330.000000
max      550.000000           895.000000             1380.000000

         Television       Golfing       Running  Volunteering
count  41098.000000  41098.000000  41098.000000  41098.000000
mean     133.943671      1.329481      0.780306      8.898900
std      140.134300     18.821230      7.582888     46.870846
min        0.000000      0.000000      0.000000      0.000000
25%       20.000000      0.000000      0.000000      0.000000
50%      105.000000      0.000000      0.000000      0.000000
75%      195.000000      0.000000      0.000000      0.000000
max     1380.000000    600.000000    505.000000   1127.000000
```

[8]:
```
unemploy=df[df['Employment Status']== 'Unemployed']
unemploy.describe()
# min and mx of weekly earnings and weekly hours worked are 0. This is waht we␣
↪suspected
```

[8]:
```
               Age     Children  Weekly Earnings  Weekly Hours Worked  \
count  3278.000000  3278.000000           3278.0               3278.0
mean     36.132703     1.134838              0.0                  0.0
std      15.835815     1.231813              0.0                  0.0
min      15.000000     0.000000              0.0                  0.0
25%      21.000000     0.000000              0.0                  0.0
50%      34.000000     1.000000              0.0                  0.0
75%      48.000000     2.000000              0.0                  0.0
max      85.000000    11.000000              0.0                  0.0

          Sleeping     Grooming   Housework  Food & Drink Prep  \
count  3278.000000  3278.000000  3278.00000        3278.000000
mean    558.650397    37.078401    48.66870          39.481391
std     147.630273    38.693576    90.12925          57.107403
min       0.000000     0.000000     0.00000           0.000000
25%     475.000000     0.000000     0.00000           0.000000
50%     540.000000    30.000000     0.00000          15.000000
```

```
75%     642.500000    60.000000    60.00000          60.000000
max    1340.000000   450.000000   778.00000         540.000000

       Caring for Children  Playing with Children  Job Searching     Shopping  \
count          3278.000000            3278.000000    3278.000000  3278.000000
mean             35.507932              10.804454      23.356010    24.615924
std              84.910312              44.791361      76.731025    50.477668
min               0.000000               0.000000       0.000000     0.000000
25%               0.000000               0.000000       0.000000     0.000000
50%               0.000000               0.000000       0.000000     0.000000
75%              16.000000               0.000000       0.000000    30.000000
max             800.000000             720.000000     983.000000   879.000000

       Eating and Drinking  Socializing & Relaxing   Television      Golfing  \
count          3278.000000             3278.000000  3278.000000  3278.000000
mean             60.554301              340.712325   195.022270     0.697071
std              52.407303              203.668876   176.454361    12.515260
min               0.000000                0.000000     0.000000     0.000000
25%              30.000000              180.000000    60.000000     0.000000
50%              50.000000              318.000000   150.000000     0.000000
75%              80.000000              480.000000   287.750000     0.000000
max             670.000000             1035.000000   981.000000   305.000000

           Running  Volunteering
count  3278.000000   3278.000000
mean      0.885601     10.648261
std       7.971380     58.254232
min       0.000000      0.000000
25%       0.000000      0.000000
50%       0.000000      0.000000
75%       0.000000      0.000000
max     150.000000    891.000000
```

[9]:
```python
notinlabor=df[df['Employment Status']== 'Not in labor force']
# min and max no of weekly hours are zero, this is what we have suspected
notinlabor.describe()
```

[9]:
```
                Age      Children  Weekly Earnings  Weekly Hours Worked  \
count  19630.000000  19630.000000          19630.0              19630.0
mean      56.064340      0.668416              0.0                  0.0
std       20.817306      1.130145              0.0                  0.0
min       15.000000      0.000000              0.0                  0.0
25%       39.000000      0.000000              0.0                  0.0
50%       62.000000      0.000000              0.0                  0.0
75%       73.000000      1.000000              0.0                  0.0
max       85.000000     12.000000              0.0                  0.0

           Sleeping      Grooming     Housework  Food & Drink Prep  \
```

```
count  19630.000000  19630.000000  19630.000000        19630.000000
mean     545.056495     38.921039     50.903413           44.308202
std      141.522954     39.615298     91.926568           62.668534
min        0.000000      0.000000      0.000000            0.000000
25%      465.000000      0.000000      0.000000            0.000000
50%      540.000000     30.000000      0.000000           20.000000
75%      617.750000     60.000000     62.000000           65.000000
max     1423.000000    550.000000   1405.000000          870.000000

       Caring for Children  Playing with Children  Job Searching  \
count         19630.000000           19630.000000   19630.000000
mean             27.308966               7.493327       0.583393
std              79.564257              37.563421      11.365300
min               0.000000               0.000000       0.000000
25%               0.000000               0.000000       0.000000
50%               0.000000               0.000000       0.000000
75%               0.000000               0.000000       0.000000
max             950.000000             840.000000     550.000000

           Shopping  Eating and Drinking  Socializing & Relaxing  \
count  19630.000000         19630.000000            19630.000000
mean      24.559705            71.884768              388.659959
std       49.353265            52.747263              222.299631
min        0.000000             0.000000                0.000000
25%        0.000000            30.000000              220.000000
50%        0.000000            60.000000              370.000000
75%       30.000000            95.000000              539.000000
max      640.000000           600.000000             1434.000000

         Television       Golfing       Running  Volunteering
count  19630.000000  19630.000000  19630.000000  19630.000000
mean     225.531228      1.316760      0.455884     11.649109
std      200.891476     18.784825      6.965441     53.882580
min        0.000000      0.000000      0.000000      0.000000
25%       64.000000      0.000000      0.000000      0.000000
50%      180.000000      0.000000      0.000000      0.000000
75%      330.000000      0.000000      0.000000      0.000000
max     1313.000000    490.000000    330.000000   1100.000000
```

[10]:
```
# lets check quantile for each of the data columns

df.quantile([0.1,0.90,0.98])
```

[10]:
```
       Age  Children  Weekly Earnings  Weekly Hours Worked  Sleeping  \
0.10  24.0       0.0              0.0                  0.0     370.0
0.90  71.0       2.0           1346.0                 50.0     690.0
0.98  80.0       4.0           2500.0                 65.0     840.0
```

```
      Grooming  Housework  Food & Drink Prep  Caring for Children  \
0.10       0.0        0.0                0.0                  0.0
0.90      90.0      135.0               97.0                110.0
0.98     130.0      300.0              186.0                295.0

      Playing with Children  Job Searching  Shopping  Eating and Drinking  \
0.10                    0.0            0.0       0.0                 15.0
0.90                    0.0            0.0      85.0                133.5
0.98                  135.0            0.0     180.0                210.0

      Socializing & Relaxing  Television  Golfing  Running  Volunteering
0.10                    55.0         0.0      0.0      0.0           0.0
0.90                   583.0       390.0      0.0      0.0           0.0
0.98                   795.0       650.0      0.0      0.0         175.0
```

[11]:
```
# lets calculate number of records where individul is employeed but working for␣
↪0 hours but earrnings are not equal to 0
#(gapminder['year']== 1967) & (gapminder['pop']>1000000) & (df['Weekly␣
↪Earnings']!=0)
records1=df.loc[(df['Employment Status'] == 'Employed') & (df['Weekly Hours␣
↪Worked']==0) & (df['Weekly Earnings']!=0)]
print (records1.shape)

# There are people who have worked for 160 and 120 and 110 hours of the week
# Is it possible?? No as a week has only 24*7= 168 hours!!! and if we assume␣
↪person has wokrd for 7 days a week for whole 24 hours then total hours will␣
↪not exceed moe than 120
# we need to drop these records
# from data of labor buero avg hours for US are Weekdays average: 8.5 hours and␣
↪weekend 5.4
# for us lets assume 9 and 6 respectively which will total up 57

records2=df.loc[(df['Employment Status']== 'Employed') & (df['Weekly Hours␣
↪Worked']>60)]
records2.shape
```

```
(1533, 23)
```

[11]: `(1710, 23)`

[12]:
```
df=df.drop(records1.index, inplace = False, axis=0)
```

[13]:
```
df=df.drop(records2.index,inplace=False,axis=0)
```

[14]:
```
df.shape
```

[14]: `(60763, 23)`

Total hours in a day cant be go beyond 24 hours, thus there is certain chances of double counting of hours in some of the independant variables.

We need to plot correlation plot to see which variables are correlated

[15]:
```
total=df.loc[df['Total']>24]

total.shape
```

```
    ␣
↪---------------------------------------------------------------------------

        KeyError                                  Traceback (most recent call␣
↪last)

        ~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in␣
↪get_loc(self, key, method, tolerance)
       2656             try:
    -> 2657                 return self._engine.get_loc(key)
       2658             except KeyError:


        pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()


        pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()


        pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()


        pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()


        KeyError: 'Total'


    During handling of the above exception, another exception occurred:


        KeyError                                  Traceback (most recent call␣
↪last)

        <ipython-input-15-3262e4f499a6> in <module>
    ----> 1 total=df.loc[df['Total']>24]
          2
          3 total.shape
```

```
    ~\Anaconda3\lib\site-packages\pandas\core\frame.py in __getitem__(self,␣
↪key)
   2925             if self.columns.nlevels > 1:
   2926                 return self._getitem_multilevel(key)
-> 2927             indexer = self.columns.get_loc(key)
   2928             if is_integer(indexer):
   2929                 indexer = [indexer]


    ~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in␣
↪get_loc(self, key, method, tolerance)
   2657                 return self._engine.get_loc(key)
   2658             except KeyError:
-> 2659                 return self._engine.get_loc(self.
↪_maybe_cast_indexer(key))
   2660         indexer = self.get_indexer([key], method=method,␣
↪tolerance=tolerance)
   2661         if indexer.ndim > 1 or indexer.size > 1:


    pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()


    pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()


    pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()


    pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()


    KeyError: 'Total'
```

[16]:
```
# lets check for sleeping patterns
df['Sleeping'].describe()
```

[16]:
```
count    60763.000000
mean       524.533762
std        134.789819
min          0.000000
25%        450.000000
50%        510.000000
```

```
75%          600.000000
max         1423.000000
Name: Sleeping, dtype: float64
```

[17]:
```
# its deemed impossible for individual to slepp for more than 14 hours a day␣
↪and less than 4 hours a day

sleep=df.loc[(df['Sleeping']<241) | (df['Sleeping']>840)]
sleep.shape
```

[17]:
```
(2253, 23)
```

[18]:
```
df=df.drop(sleep.index, inplace=False, axis=0)
```

[19]:
```
# Grooming
print(df['Grooming'].describe())
print()
print (df.Grooming.quantile([0.95,0.98]))
```

```
count    58510.000000
mean        40.795437
std         36.347839
min          0.000000
25%         15.000000
50%         33.000000
75%         60.000000
max       1043.000000
Name: Grooming, dtype: float64

0.95    105.0
0.98    130.0
Name: Grooming, dtype: float64
```

[20]:
```
df=df.drop((df.loc[df['Grooming']>130]).index,inplace=False, axis=0)

df.shape
```

[20]:
```
(57358, 23)
```

[21]:
```
#Housework
print(df.Housework.describe())
print()
print(df.Housework.quantile([0.80,0.90,0.98]))
```

```
count    57358.000000
mean        42.617909
std         83.434055
min          0.000000
25%          0.000000
50%          0.000000
```

```
75%          60.000000
max        1030.000000
Name: Housework, dtype: float64

0.80     75.0
0.90    140.3
0.98    300.0
Name: Housework, dtype: float64
```

[22]:
```
df=df.drop((df.loc[df.Housework>(df['Housework'].quantile(.98))]).
 ↪index,inplace=False,axis=0)
print()
print(df.shape)
```

```
(56213, 23)
```

[23]:
```
## Food & Drink Prep

print(df['Food & Drink Prep'].describe())
print()
print(df['Food & Drink Prep'].quantile([0.85,0.95,0.98]))
df=df.drop((df.loc[df['Food & Drink Prep']>185]).index, inplace=False, axis=0)
print()
print(df.shape)
```

```
count    56213.000000
mean        35.013520
std         53.415314
min          0.000000
25%          0.000000
50%         15.000000
75%         55.000000
max        755.000000
Name: Food & Drink Prep, dtype: float64

0.85     76.0
0.95    135.0
0.98    190.0
Name: Food & Drink Prep, dtype: float64

(55066, 23)
```

[24]:
```
# its is quite possible for people to search jobs for 4 hours in a day.

print(df['Job Searching'].value_counts().nlargest(10))
print()
```

```
print(df['Job Searching'].describe())

print(df['Job Searching'].quantile([0.85,0.95,0.99]))
df=df.drop((df.loc[df['Job Searching']>240]).index, inplace=False, axis=0)
print()
print(df.shape)
```

```
0      54310
60       105
120       90
30        75
90        42
180       40
45        28
20        26
240       23
150       18
Name: Job Searching, dtype: int64

count    55066.000000
mean         1.836760
std         21.709308
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max        983.000000
Name: Job Searching, dtype: float64
0.85     0.0
0.95     0.0
0.99    58.7
Name: Job Searching, dtype: float64

(54957, 23)
```

[26]:
```
# shopping

print(df.Shopping.value_counts().nlargest(10))
print()
print(df.Shopping.describe())
print()
print(df.Shopping.quantile([0.95,0.98]))

df.shape
```

```
0      31079
60      2232
```

```
30      2218
10      2040
5       1813
15      1646
20      1453
45      1039
90       977
120      936
Name: Shopping, dtype: int64

count    54957.000000
mean        25.303292
std         49.625488
min          0.000000
25%          0.000000
50%          0.000000
75%         30.000000
max        640.000000
Name: Shopping, dtype: float64

0.95    120.0
0.98    180.0
Name: Shopping, dtype: float64
```

[26]:
```
(54957, 23)
```

[27]:
```
df=df.drop((df.loc[df['Running']>120]).index, inplace=False, axis=0)
```

[28]:
```
df.shape
```

[28]:
```
(54932, 23)
```

[29]:
```
# Running
print(df.Running.value_counts().nlargest(10))
print()
print(df.Running.describe())
print()
print(df.Running.quantile([0.95,0.98]))
```

```
0      54254
60       181
30       136
45        87
40        41
90        34
120       31
20        28
25        23
35        15
```

```
Name: Running, dtype: int64

count    54932.000000
mean         0.621477
std          6.249482
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max        120.000000
Name: Running, dtype: float64

0.95    0.0
0.98    0.0
Name: Running, dtype: float64
```

[30]:

```
#House work


print(df['Housework'].value_counts().nlargest(10))
print()

print(df['Housework'].describe())

print(df['Housework'].quantile([0.85,0.95,0.98]))
df=df.drop((df.loc[df['Housework']>240]).index, inplace=False, axis=0)
print()
print(df.shape)
```

```
0      33333
60      2549
120     1959
30      1956
90      1206
15      1149
10      1110
20       939
180      813
45       711
Name: Housework, dtype: int64

count    54932.000000
mean        34.144633
std         61.689554
min          0.000000
25%          0.000000
50%          0.000000
```

```
75%           45.000000
max          300.000000
Name: Housework, dtype: float64
0.85     90.0
0.95    180.0
0.98    240.0
Name: Housework, dtype: float64

(54147, 23)
```

[31]:

```python
# Eating and Drinking
print(df['Eating and Drinking'].value_counts().nlargest(10))
print()

print(df['Eating and Drinking'].describe())

print(df['Eating and Drinking'].quantile([0.85,0.95,0.98]))
df=df.drop((df.loc[df['Eating and Drinking']>210]).index, inplace=False, axis=0)
print()
print(df.shape)
```

```
60     5596
30     5526
90     3604
45     2859
75     2419
120    2330
20     2228
50     2028
15     2000
0      1959
Name: Eating and Drinking, dtype: int64

count    54147.000000
mean        69.569856
std         52.393352
min          0.000000
25%         30.000000
50%         60.000000
75%         90.000000
max        765.000000
Name: Eating and Drinking, dtype: float64
0.85    120.0
0.95    165.0
0.98    210.0
Name: Eating and Drinking, dtype: float64

(53236, 23)
```

[32]:
```python
# Food & Drink Prep

print(df['Food & Drink Prep'].value_counts().nlargest(10))
print()

print(df['Food & Drink Prep'].describe())

print(df['Food & Drink Prep'].quantile([0.85,0.95,0.98]))
df=df.drop((df.loc[df['Food & Drink Prep']>150]).index, inplace=False, axis=0)
print()
print(df.shape)
```

```
0     22935
30     3861
60     2762
15     2129
20     1917
10     1724
45     1702
90     1231
5      1123
40     1033
Name: Food & Drink Prep, dtype: int64

count    53236.000000
mean        30.075325
std         40.422781
min          0.000000
25%          0.000000
50%         10.000000
75%         50.000000
max        185.000000
Name: Food & Drink Prep, dtype: float64
0.85     70.0
0.95    120.0
0.98    150.0
Name: Food & Drink Prep, dtype: float64

(52372, 23)
```

[33]:
```python
# Grooming



print(df['Grooming'].value_counts().nlargest(10))
print()

print(df['Grooming'].describe())
```

```
print(df['Grooming'].quantile([0.85,0.95,0.98]))
df=df.drop((df.loc[df['Food & Drink Prep']>115]).index, inplace=False, axis=0)
print()
print(df.shape)
```

```
0      11023
30      7375
60      5918
45      3758
20      2515
15      2162
90      1980
40      1960
75      1591
50      1521
Name: Grooming, dtype: int64

count    52372.000000
mean        38.733808
std         31.075133
min          0.000000
25%         15.000000
50%         32.000000
75%         60.000000
max        130.000000
Name: Grooming, dtype: float64
0.85     70.0
0.95     95.0
0.98    115.0
Name: Grooming, dtype: float64

(50155, 23)
```

[34]:

```
# Golfing

print(df['Golfing'].value_counts().nlargest(10))
print()

print(df['Golfing'].describe())

print(df['Golfing'].quantile([0.85,0.95,0.98]))
df=df.drop((df.loc[df['Golfing']>300]).index, inplace=False, axis=0)
print()
print(df.shape)
```

```
0      49800
240       29
```

```
60       26
120      26
270      22
150      20
300      19
90       12
30        9
210       9
Name: Golfing, dtype: int64

count    50155.000000
mean         1.451760
std         19.604608
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max        600.000000
Name: Golfing, dtype: float64
0.85    0.0
0.95    0.0
0.98    0.0
Name: Golfing, dtype: float64

(50088, 23)
```

[35]:
```
df.columns
```

[35]:
```
Index(['Education Level', 'Age', 'Age Range', 'Employment Status', 'Gender',
       'Children', 'Weekly Earnings', 'Year', 'Weekly Hours Worked',
       'Sleeping', 'Grooming', 'Housework', 'Food & Drink Prep',
       'Caring for Children', 'Playing with Children', 'Job Searching',
       'Shopping', 'Eating and Drinking', 'Socializing & Relaxing',
       'Television', 'Golfing', 'Running', 'Volunteering'],
      dtype='object')
```

[36]:
```
# Playing with Chidlren
print(df['Playing with Children'].value_counts().nlargest(10))
print()

print(df['Playing with Children'].describe())

print(df['Playing with Children'].quantile([0.85,0.95,0.98]))
#df=df.drop((df.loc[df['Playing with Children']>143]).index, inplace=False,␣
 ↪axis=0)
print()
print(df.shape)
```

```
0      46105
```

```
60       675
30       423
120      382
90       279
45       183
180      140
20       103
150      100
75        87
Name: Playing with Children, dtype: int64

count    50088.000000
mean         8.764435
std         39.896626
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max        840.000000
Name: Playing with Children, dtype: float64
0.85      0.0
0.95     60.0
0.98    140.0
Name: Playing with Children, dtype: float64

(50088, 23)
```

[37]:
```python
#Socializing & Relaxing

print(df['Socializing & Relaxing'].value_counts().nlargest(10))
print()

print(df['Socializing & Relaxing'].describe())

print(df['Socializing & Relaxing'].quantile([0.85,0.95,0.98]))
#df=df.drop((df.loc[df['Socializing & Relaxing']>810]).index, inplace=False,␣
 ↪axis=0)
print()
print(df.shape)
```

```
0      1955
120    1486
60     1203
180    1126
150     889
90      884
240     858
```

```
210     747
30      719
300     682
Name: Socializing & Relaxing, dtype: int64

count    50088.000000
mean       300.313109
std        209.028442
min          0.000000
25%        135.000000
50%        260.000000
75%        430.000000
max       1162.000000
Name: Socializing & Relaxing, dtype: float64
0.85    535.0
0.95    700.0
0.98    805.0
Name: Socializing & Relaxing, dtype: float64

(50088, 23)
```

[38]:

```
# Television

print(df['Television'].value_counts().nlargest(10))
print()

print(df['Television'].describe())

print(df['Television'].quantile([0.85,0.95,0.98]))
#df=df.drop((df.loc[df['Television']>615]).index, inplace=False, axis=0)
print()
print(df.shape)
```

```
0      9357
120    2926
60     2813
90     1754
30     1542
180    1430
150    1229
240     970
210     772
45      572
Name: Television, dtype: int64

count    50088.000000
mean       172.490836
```

```
std         172.064565
min           0.000000
25%          45.000000
50%         120.000000
75%         245.000000
max        1162.000000
Name: Television, dtype: float64
0.85    335.0
0.95    529.0
0.98    661.0
Name: Television, dtype: float64

(50088, 23)
```

[39]:

```
#Volunteering

print(df['Volunteering'].value_counts().nlargest(10))
print()

print(df['Volunteering'].describe())

print(df['Volunteering'].quantile([0.85,0.95,0.98]))
#df=df.drop((df.loc[df['Volunteering']>180]).index, inplace=False, axis=0)
print()
print(df.shape)
```

```
0      46254
60       325
120      245
30       229
90       183
15       169
20       136
180      134
10       124
45        97
Name: Volunteering, dtype: int64

count    50088.000000
mean        10.537474
std         51.585756
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max       1020.000000
Name: Volunteering, dtype: float64
```

```
0.85      0.0
0.95     60.0
0.98    180.0
Name: Volunteering, dtype: float64

(50088, 23)
```

[40]:
```
file=df.to_csv('file.csv')
```

[41]:
```
#Finding the different yearly median for weekly earnings among Employed

employeed=df.loc[df['Employment Status']=='Employed']

employeed.head()
```

[41]:
```
   Education Level  Age Age Range Employment Status  Gender  Children  \
1         Bachelor   42     40-49          Employed  Female         2
2           Master   47     40-49          Employed    Male         0
6      High School   46     40-49          Employed    Male         0
7         Bachelor   24     20-29          Employed  Female         0
13     High School   22     20-29          Employed  Female         2

    Weekly Earnings  Year  Weekly Hours Worked  Sleeping  ...  \
1              1480  2005                   40       500  ...
2               904  2005                   40       480  ...
6               700  2005                   40       445  ...
7               442  2005                   45       435  ...
13                0  2005                    0       430  ...

    Caring for Children  Playing with Children  Job Searching  Shopping  \
1                   365                     20              0       120
2                     0                      0              0        15
6                     0                      0              0         0
7                     0                      0              0         0
13                  147                      0              0        80

    Eating and Drinking  Socializing & Relaxing  Television  Golfing  Running  \
1                    40                      15          15        0        0
2                    85                     214         199        0        0
6                    60                     165         150        0        0
7                     0                       0           0        0        0
13                   99                     313         120        0        0

    Volunteering
1              0
2              0
6              0
7              0
```

```
13              0

[5 rows x 23 columns]
```

[5]:
```
y_2012=df.loc[df['Year']==2012]
```

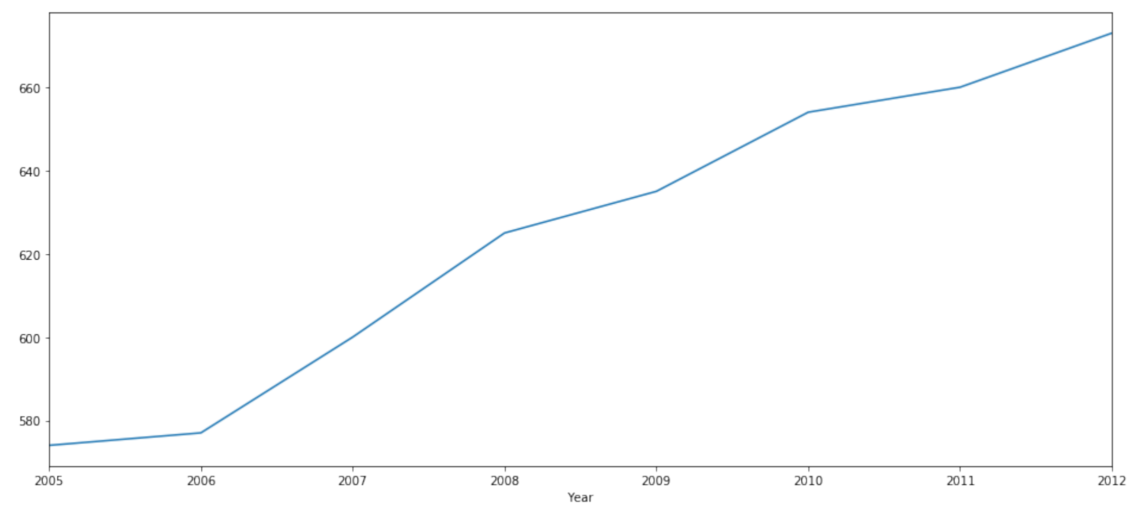
[44]:
```
df['Employment Status'].value_counts()
```

[44]:
```
Employed              32534
Not in labor force    15070
Unemployed             2484
Name: Employment Status, dtype: int64
```

[51]:
```
employeed['Year'].value_counts()
```

[51]:
```
2005    4365
2007    4211
2008    4185
2006    4177
2009    4027
2012    3893
2010    3864
2011    3812
Name: Year, dtype: int64
```

[52]:
```
d=employeed.groupby(['Year'])['Weekly Earnings'].median().plot(figsize=(16,7))
```



[53]:
```
med=employeed.groupby(['Year'])['Weekly Earnings'].median()

for year in employeed['Year'].unique():
    print(f' For the year {year} median weekly earnings were {med[year]}\n')
```

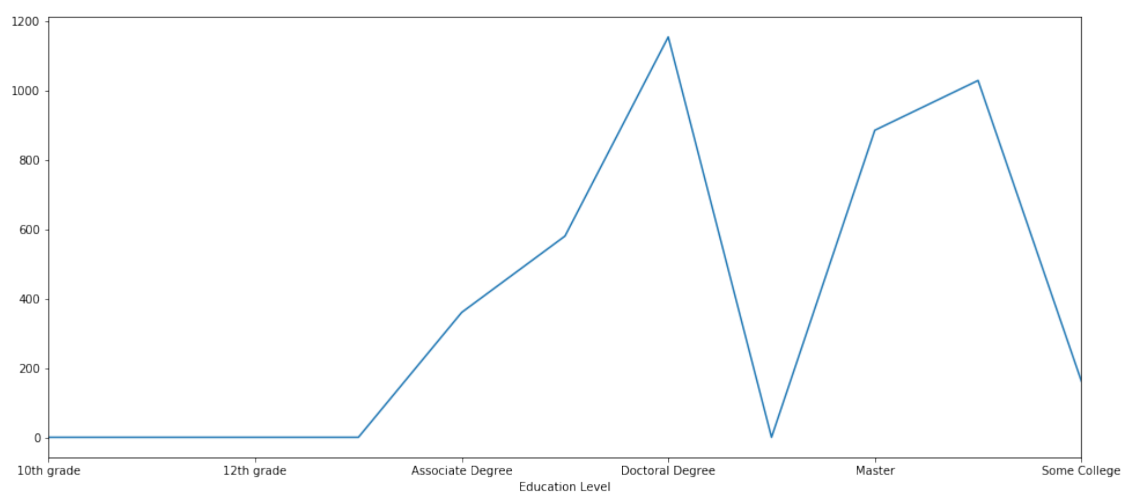
```
 For the year 2005 median weekly earnings were 574
```

```
For the year 2006 median weekly earnings were 577

For the year 2007 median weekly earnings were 600

For the year 2008 median weekly earnings were 625

For the year 2009 median weekly earnings were 635

For the year 2010 median weekly earnings were 654

For the year 2011 median weekly earnings were 660

For the year 2012 median weekly earnings were 673
```

[ ]:
```
# Analyze data for year 2005
```

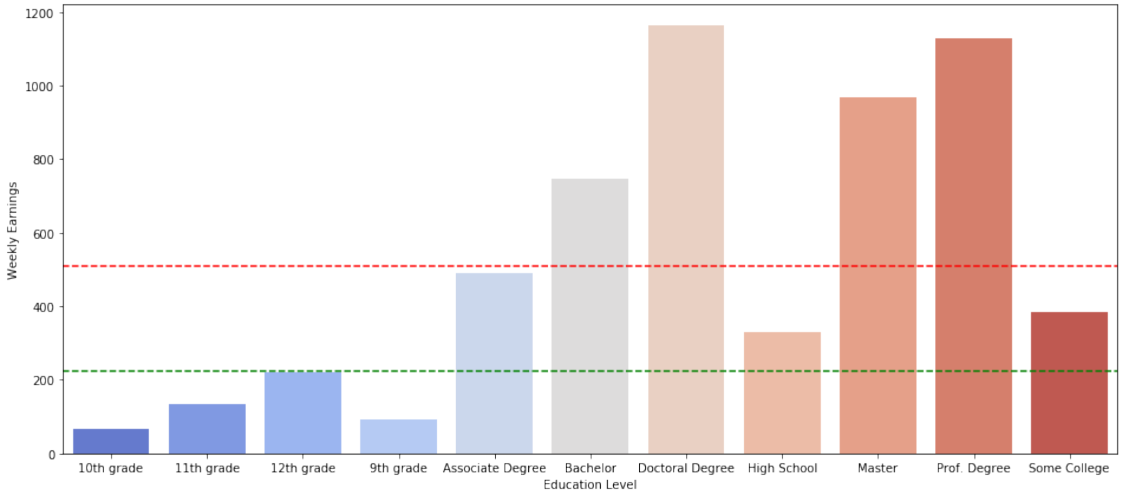
[42]:
```
d=y_2012.groupby(['Education Level'])['Weekly Earnings'].median().
↪plot(figsize=(16,7))
d
```

[42]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x209ffc64eb8>
```



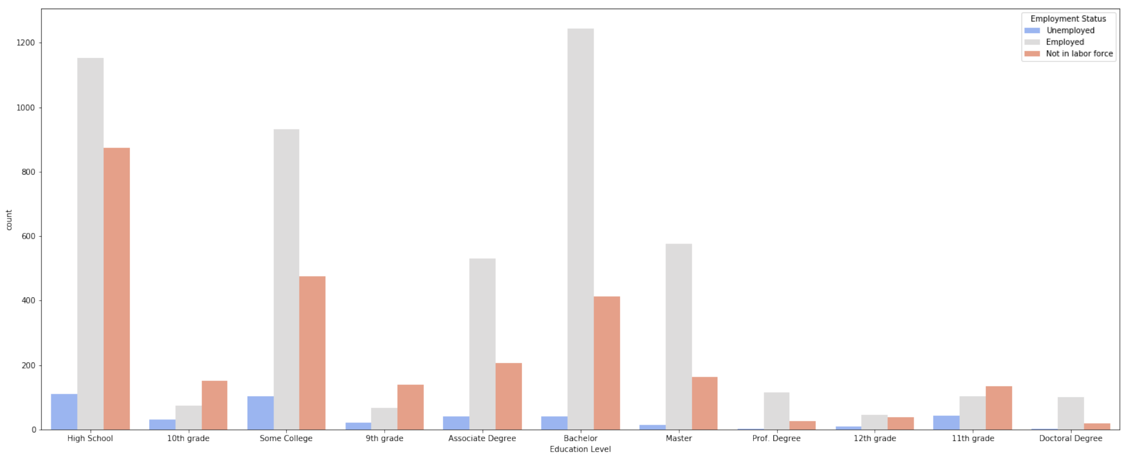
[54]:
```
mlt.subplots(figsize=(16,7))
d=y_2012.groupby(['Education Level'])['Weekly Earnings'].mean()

snr.barplot(x=d.index,y=d, palette='coolwarm')
mean=y_2012['Weekly Earnings'].mean()
median=y_2012['Weekly Earnings'].median()
mlt.axhline(mean, color='r', linestyle='--')
mlt.axhline(median, color='g', linestyle='--')
```

[54]: <matplotlib.lines.Line2D at 0x234c8105860>



[6]:
```
m=y_2012.groupby(['Education Level'])['Employment Status'].count()
fig,ax= mlt.subplots(nrows=1, ncols=1, figsize=(25,10))
snr.countplot(x=y_2012['Education Level'],␣
↪palette='coolwarm',hue=y_2012['Employment Status'])
```

[6]: <matplotlib.axes._subplots.AxesSubplot at 0x1e3576357f0>



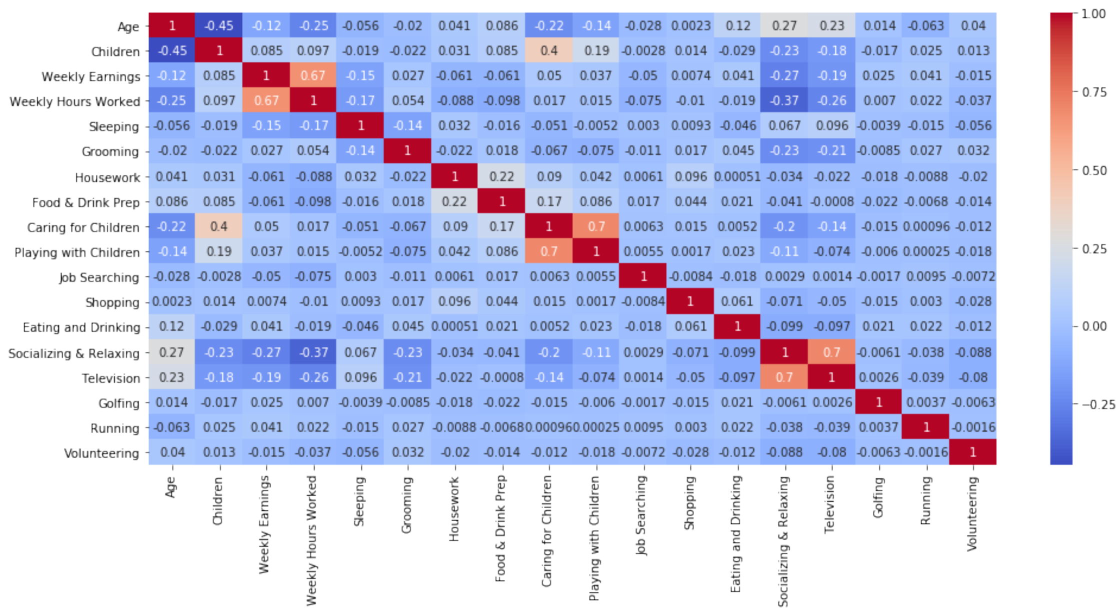
[7]:
```
m
```

[7]:
```
Education Level
10th grade          257
11th grade          280
12th grade           92
9th grade           229
```

```
Associate Degree     778
Bachelor            1697
Doctoral Degree      122
High School         2140
Master               755
Prof. Degree         142
Some College        1509
Name: Employment Status, dtype: int64
```

[59]:
```
fig,ax=mlt.subplots(nrows=1,ncols=1,figsize=(16,7))
snr.heatmap(df.corr(),cmap='coolwarm',annot=True)
```

[59]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x234ce151b70>
```



Caring for children and playing with children have high correlation of 0.7. Caring for chindren has bigger scope which will also include playing with children along with the other parenting aspects

Similarly Television is a subset of Socializing and Relaxing

Thus let's drop Television and playing with children

[60]:
```
df=df.drop(['Television','Playing with Children'],axis=1)
df.shape
```

[60]:
```
(50088, 21)
```

2 Chi Square Test

lets try to see association between between Employment Status and Gender we wil use chi square test, its used to identified whather there is a association between two categorical variables from the same sample population

2.1 Are Employment Status and Gender independant of Each Other in 2012?

null hypothesis: Independantn (p-val > 0.05) Alternate Hypothesis: There is a rcorrelation between Employment and Gender (p-val<0.5)

[61]:
```
import scipy.stats as stats
emp_tab=pd.crosstab(y_2012['Employment Status'],y_2012['Gender'])

print(emp_tab)
print()
print(emp_tab.shape)
```

```
Gender              Female  Male
Employment Status
Employed              1906  1987
Not in labor force    1267   773
Unemployed             146   158

(3, 2)
```

assumptions of chi-square: - The data in the cells should be frequencies, or counts of cases rather than percentages or some other transformation of the data.

- The levels (or categories) of the variables are mutually exclusive. That is, a particular subject fits into one and only one level of each of the variables.
- The value of the cell expecteds should be 5 or more in at least 80% of the cells, and no cell should have an expected of less than one

[62]:
```
observed_val=emp_tab.values
observed_val
```

[62]:
```
array([[1906, 1987],
       [1267,  773],
       [ 146,  158]], dtype=int64)
```

[63]:
```
val=stats.chi2_contingency(observed_val)
```

[64]:
```
no_of_rows=len(emp_tab.iloc[0:,0])
no_of_cols=len(emp_tab.iloc[0,0:])
ddof=(no_of_rows-1)*(no_of_cols-1)
print(f' Degree of freedom is : {ddof}')
alpha=0.05
```

```
 Degree of freedom is : 2
```

[65]:
```
# 3 because we are interested in array in val which is at 3rd index strats from␣
↪0
print(val)
expected_val=val[3]
```

```
(96.40121307655733, 1.166118152880802e-21, 2, array([[2071.64774731,
1821.35225269],
       [1085.57960558,  954.42039442],
       [ 161.77264711,  142.22735289]]))
```

[66]:
```
from scipy.stats import chi2
```

[67]:
```
chi_square=sum([(o-e)**2./e for o,e in zip(observed_val,expected_val)])
chi_stats=chi_square[0]+chi_square[1]
print('Chi Square Statistic :',chi_stats)
```

```
Chi Square Statistic : 96.40121307655733
```

[68]:
```
critical_value=chi2.ppf(q=1-alpha,df=ddof)
print('Critical Value:', critical_value)
```

```
Critical Value: 5.991464547107979
```

[69]:
```
# p value
p_val=1-chi2.cdf(x=chi_stats,df=ddof)
print('P-val:',p_val)
print()
print('Significance Level:', alpha)
print()
print(f'Degree of freedom is : {ddof}')
```

```
P-val: 0.0

Significance Level: 0.05

Degree of freedom is : 2
```

[70]:
```
if p_val>=alpha:
    print('There is no relation between Employment Status and Gender')
else:
    print('There is a relation between Employment Status and Gender')
```

```
There is a relation between Employment Status and Gender
```

[71]:
```
#Lets Explore the Gender and Employment Status

y_2012.groupby(['Employment Status'])['Gender'].count()
```

[71]:
```
Employment Status
Employed              3893
Not in labor force    2040
Unemployed             304
Name: Gender, dtype: int64
```

[72]:
```
y_2012.groupby(['Education Level'])['Employment Status'].count()
```

[72]:
```
Education Level
10th grade          203
11th grade          220
12th grade           61
9th grade           181
Associate Degree    599
Bachelor           1322
Doctoral Degree      91
High School        1655
Master              624
Prof. Degree        107
Some College       1174
Name: Employment Status, dtype: int64
```

2.2 Is there correlation exists between Education level and Employment status in 2012?

[45]:
```
import scipy.stats as stats
emp_tab=pd.crosstab(y_2012['Employment Status'],y_2012['Education Level'])

print(emp_tab)
print()
print(emp_tab.shape)
```

```
Education Level     10th grade  11th grade  12th grade  9th grade  \
Employment Status
Employed                    54          81          35         52
Not in labor force         124         108          18        111
Unemployed                  25          31           8         18

Education Level     Associate Degree  Bachelor  Doctoral Degree  High School  \
Employment Status
Employed                         418       982               74          902
Not in labor force               155       312               16          671
Unemployed                        26        28                1           82
```

```
Education Level      Master  Prof. Degree  Some College
Employment Status
Employed                485            84           726
Not in labor force      130            22           373
Unemployed                9             1            75
```

```
(3, 11)
```

assumptions of chi-square: - The data in the cells should be frequencies, or counts of cases rather than percentages or some other transformation of the data.

- The levels (or categories) of the variables are mutually exclusive. That is, a particular subject fits into one and only one level of each of the variables.
- The value of the cell expecteds should be 5 or more in at least 80% of the cells, and no cell should have an expected of less than one

[74]:
```
# it does satisfies the conditions
observed_val=emp_tab.values
observed_val
```

[74]:
```
array([[ 54,  81,  35,  52, 418, 982,  74, 902, 485,  84, 726],
       [124, 108,  18, 111, 155, 312,  16, 671, 130,  22, 373],
       [ 25,  31,   8,  18,  26,  28,   1,  82,   9,   1,  75]],
      dtype=int64)
```

[75]:
```
val=stats.chi2_contingency(observed_val)
print(val)
expected_val=val[3]
```

```
(534.920074221628, 1.3930165578725052e-100, 20, array([[ 126.70819304,
137.31922399,   38.07487574,  112.97627064,
         373.88279622,  825.1637005 ,   56.80022447, 1033.01507135,
         389.48725349,   66.78707712,  732.78531345],
       [  66.3973064 ,   71.95767196,   19.95189995,   59.2015392 ,
         195.92111592,  432.4001924 ,   29.76430976,  541.31794132,
         204.0981241 ,   34.997595  ,  383.99230399],
       [   9.89450056,   10.72310406,    2.97322431,    8.82219016,
          29.19608786,   64.4361071 ,    4.43546577,   80.66698733,
          30.41462241,    5.21532788,   57.22238256]]))
```

[76]:
```
from scipy.stats import chi2
```

[77]:
```
chi_square=sum([(o-e)**2./e for o,e in zip(observed_val,expected_val)])
chi_stats=chi_square[0]+chi_square[1]
print('Chi Square Statistic :',chi_stats)
```

```
Chi Square Statistic : 194.2496161798971
```

[78]:
```python
critical_value=chi2.ppf(q=1-alpha,df=ddof)
print('Critical Value:', critical_value)
```

```
Critical Value: 5.991464547107979
```

[79]:
```python
p_val=1-chi2.cdf(x=chi_stats,df=ddof)
print('P-val:',p_val)
print()
print('Significance Level:', alpha)
print()
print(f'Degree of freedom is : {ddof}')
```

```
P-val: 0.0

Significance Level: 0.05

Degree of freedom is : 2
```

[80]:
```python
if p_val>=alpha:
    print('There is no relation between Employment Status and Education Level')
else:
    print('There is a relation between Employment Status and Education Level')
```

```
There is a relation between Employment Status and Education Level
```

3 Anova

- One way Annova: analyzing the test score of a class based on gender
- Two Way Anova: analyzing the test score of a class based on gender and age. Here test score is a dependent variable and gender and age are the independent variables. Two-way ANOVA can be used to find the relationship between these dependent and independent variables

[ ]:
```python
y_2012.info()
```

[46]:
```python
# Annova
import statsmodels.api as sm
from statsmodels.formula.api import ols
```

[47]:
```python
# in Annova make sure dependant variable is converted into numbers
y_2012['Gender'] = y_2012['Gender'].replace({'Male':1, 'Female':0})
y_2012['EmpStatus']=y_2012['Employment Status'].replace({'Employed':
↪1,'Unemployed':2,'Not in labor force':3})
```

[48]:
```python
y_2012['Week_earn']=y_2012['Weekly Earnings']
```

[49]:
```python
y_2012['weekly_hours']=y_2012['Weekly Hours Worked']
```

[50]:
```
y_2012['Eating']=y_2012['Eating and Drinking']
```

[51]:
```
y_2012['child']=y_2012['Caring for Children']
```

[52]:
```
y_2012.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6237 entries, 56005 to 64005
Data columns (total 28 columns):
Education Level          6237 non-null object
Age                      6237 non-null int64
Age Range                6237 non-null object
Employment Status        6237 non-null object
Gender                   6237 non-null int64
Children                 6237 non-null int64
Weekly Earnings          6237 non-null int64
Year                     6237 non-null object
Weekly Hours Worked      6237 non-null int64
Sleeping                 6237 non-null int64
Grooming                 6237 non-null int64
Housework                6237 non-null int64
Food & Drink Prep        6237 non-null int64
Caring for Children      6237 non-null int64
Playing with Children    6237 non-null int64
Job Searching            6237 non-null int64
Shopping                 6237 non-null int64
Eating and Drinking      6237 non-null int64
Socializing & Relaxing   6237 non-null int64
Television               6237 non-null int64
Golfing                  6237 non-null int64
Running                  6237 non-null int64
Volunteering             6237 non-null int64
EmpStatus                6237 non-null int64
Week_earn                6237 non-null int64
weekly_hours             6237 non-null int64
Eating                   6237 non-null int64
child                    6237 non-null int64
dtypes: int64(24), object(4)
memory usage: 1.4+ MB
```

3.1 Effect of gender male and female on Shopping: One way Anova

3.2 Effect of gender male and female on Shopping and EmpStatus: Two way Anova

[55]:
```
# two way annova is used to comapre multiple dependant variables
# make sure independant variable is converted into numbers
#y_2005['Gender'] = y_2005['Gender'].replace({'Male':1, 'Female':0})
```

```python
#y_2005['EmpStatus']=y_2005['Employment Status'].replace({'Employed':
 ↪1,'Unemployed':2,'Not in labor force':3})
# effect on gender male and female on Shopping one way
#C(Gender):C(EmpStatus) insteraction term
print('Effect on gender male and female on Shopping: One way Anova')
print()
model1 = ols("Shopping ~ C(Gender)", data=y_2012).fit()
anova_table1 = sm.stats.anova_lm(model1, typ=1)
print(anova_table1)

model2 = ols("Shopping ~ C(EmpStatus)", data=y_2012).fit()
anova_table2 = sm.stats.anova_lm(model2, typ=1)

print(anova_table2)
print()
print()

# effect on gender male and female on Shopping and EmpStatus Two way
print('Effect on gender and employee status Shopping and interaction between␣
 ↪gender and emp status: Two way Anova')
print()
model3 = ols("Shopping ~ +C(Gender)+C(EmpStatus)+C(Gender):C(EmpStatus)",␣
 ↪data=y_2012).fit()
anova_table3 = sm.stats.anova_lm(model3, typ=2)
print(anova_table3)
```

```
Effect on gender male and female on Shopping: One way Anova

                df        sum_sq        mean_sq          F        PR(>F)
C(Gender)      1.0  1.278643e+05  127864.324877  58.698456  2.118145e-14
Residual    6235.0  1.358186e+07    2178.325186        NaN           NaN
                   df        sum_sq       mean_sq         F    PR(>F)
C(EmpStatus)      2.0  3.874355e+02    193.717736  0.088089  0.915681
Residual       6234.0  1.370933e+07   2199.123264       NaN       NaN


Effect on gender and employee status Shopping and interaction between gender and
emp status: Two way Anova

                              sum_sq      df          F        PR(>F)
C(Gender)               1.316533e+05     1.0  60.453625  8.756610e-15
C(EmpStatus)            4.176444e+03     2.0   0.958886  3.833761e-01
C(Gender):C(EmpStatus)  8.074281e+03     2.0   1.853806  1.567262e-01
Residual                1.356961e+07  6231.0        NaN           NaN
```

As we can see in the annova test, shoping has no difference on the basis of emp status. That is weather person is employed or not employed there no chnage in shopping pattern

[89]:
```
model = ols("Grooming ~ C(EmpStatus)", data=y_2012).fit()
anova_table = sm.stats.anova_lm(model, typ=1)
print(anova_table)
```

```
                  df        sum_sq       mean_sq          F    PR(>F)
C(EmpStatus)     2.0  2.431334e+04  12156.668484  12.280784  0.000005
Residual      6234.0  6.170996e+06    989.893504        NaN       NaN
```

[90]:
```
model = ols("Television ~ C(EmpStatus)", data=y_2012).fit()
anova_table = sm.stats.anova_lm(model, typ=1)
print(anova_table)
```

```
                  df        sum_sq       mean_sq           F         PR(>F)
C(EmpStatus)     2.0  1.641575e+07  8.207876e+06  277.951785  2.340172e-116
Residual      6234.0  1.840891e+08  2.952986e+04         NaN            NaN
```

[91]:
```
model = ols("Running ~ +C(EmpStatus)", data=y_2012).fit()
anova_table = sm.stats.anova_lm(model, typ1=1)
print(anova_table)
```

```
                  df         sum_sq    mean_sq         F    PR(>F)
C(EmpStatus)     2.0     168.934489  84.467245  1.711378  0.180701
Residual      6234.0  307686.938527  49.356262       NaN       NaN
```

[92]:
```
model = ols("Sleeping ~ +C(EmpStatus)", data=y_2012).fit()
anova_table = sm.stats.anova_lm(model, typ1=1)
print(anova_table)
```

```
                  df        sum_sq         mean_sq          F        PR(>F)
C(EmpStatus)     2.0  1.936950e+06   968474.980662  79.039806  1.263132e-34
Residual      6234.0  7.638522e+07    12253.003070        NaN           NaN
```

[61]:
```
model = ols("Eating ~ +C(EmpStatus)", data=y_2012).fit()
anova_table = sm.stats.anova_lm(model, typ1=1)
print(anova_table)
```

```
                  df        sum_sq       mean_sq         F    PR(>F)
C(EmpStatus)     2.0  3.074952e+04  15374.759492  5.361411  0.004714
Residual      6717.0  1.926214e+07   2867.670620       NaN       NaN
```

[93]:
```
model = ols("child ~ +C(EmpStatus)", data=y_2012).fit()
anova_table = sm.stats.anova_lm(model, typ1=1)
print(anova_table)
```

```
                  df        sum_sq       mean_sq          F    PR(>F)
C(EmpStatus)     2.0  1.488218e+05  74410.911717  13.657949  0.000001
Residual      6234.0  3.396393e+07   5448.176146        NaN       NaN
```

[94]:
```
emp_2012=y_2012.loc[y_2012['EmpStatus']==1]
```

[95]:
```
emp_2012.columns
```

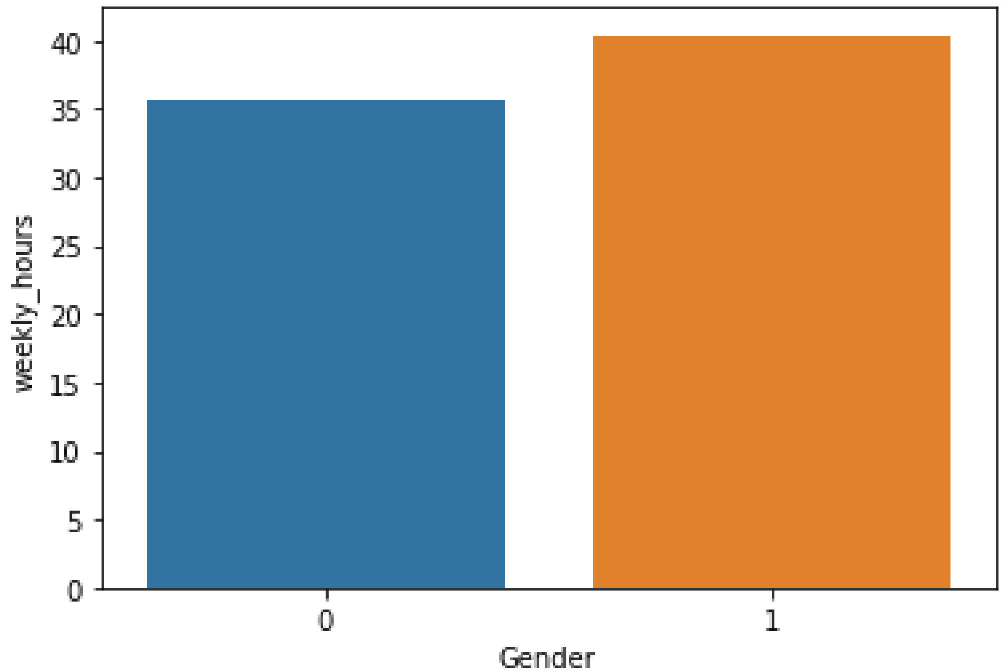
[95]:
```
Index(['Education Level', 'Age', 'Age Range', 'Employment Status', 'Gender',
       'Children', 'Weekly Earnings', 'Year', 'Weekly Hours Worked',
       'Sleeping', 'Grooming', 'Housework', 'Food & Drink Prep',
       'Caring for Children', 'Playing with Children', 'Job Searching',
       'Shopping', 'Eating and Drinking', 'Socializing & Relaxing',
       'Television', 'Golfing', 'Running', 'Volunteering', 'EmpStatus',
       'Week_earn', 'weekly_hours', 'Eating', 'child'],
      dtype='object')
```

[96]:
```
# Relation ship between weekly earnings and gender

model = ols("Week_earn ~ +C(Gender)", data=emp_2012).fit()
anova_table = sm.stats.anova_lm(model, typ1=1)
print(anova_table)

# there is disparicty between earnings between male and female
```

```
               df        sum_sq       mean_sq           F        PR(>F)
C(Gender)     1.0  5.183326e+07  5.183326e+07  111.070954  1.260308e-25
Residual   3891.0  1.815805e+09  4.666680e+05         NaN           NaN
```

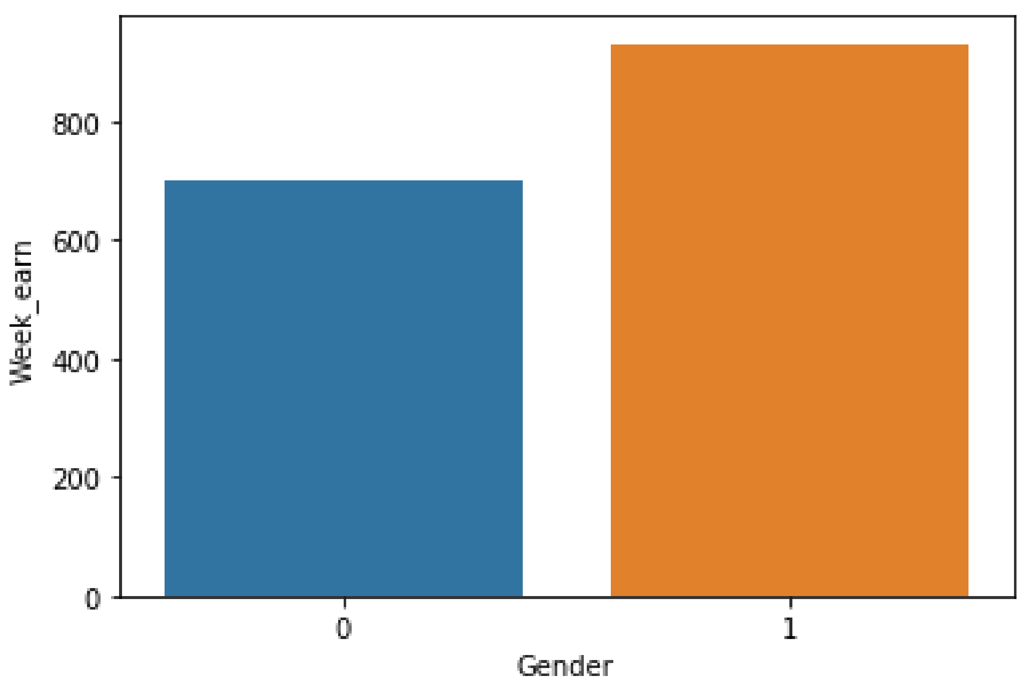
[97]:
```
d=emp_2012.groupby(['Gender'])['weekly_hours'].mean()

d
snr.barplot(x=d.index,y=d)
```

[97]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x234ccf806d8>
```



[98]:
```
d=emp_2012.groupby(['Gender'])['Week_earn'].mean()

d
snr.barplot(x=d.index,y=d)
```

[98]: <matplotlib.axes._subplots.AxesSubplot at 0x234ccfd1748>



[99]:
```
model = ols("weekly_hours ~ +C(Gender)", data=emp_2012).fit()
anova_table = sm.stats.anova_lm(model, typ1=1)
print(anova_table)
```

```
                df         sum_sq       mean_sq           F        PR(>F)
C(Gender)      1.0   20931.636664  20931.636664  139.202976  1.365398e-31
Residual    3891.0  585080.867831    150.367738         NaN           NaN
```

[ ]:
```
df.info()
```

4 Train Test Split

[67]:
```
# train Test split

# lets create train and test data

from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test=train_test_split(data.drop(['Employment␣
↪Status'],axis=1),data['Employment Status'],
                                                test_size=0.30,random_state=42)

print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
```

```
(44804, 24)
(44804,)
(19202, 24)
```

[90]:
```
output = pd.DataFrame(index=None, columns=['model','train_Rsquare',␣
↪'test_Rsquare','CV_Score'])
```

[ ]:
```
X_train.columns
```

[68]:
```
from sklearn.model_selection import train_test_split
# from feature-engine
from feature_engine import missing_data_imputers as mdi
# for one hot encoding with feature-engine
from feature_engine.categorical_encoders import OneHotCategoricalEncoder
from feature_engine.categorical_encoders import RareLabelCategoricalEncoder
from sklearn.pipeline import Pipeline as skpipe
from sklearn.preprocessing import MinMaxScaler
```

[69]:
```
from sklearn.preprocessing import MinMaxScaler
time=skpipe([

('one hot encoding',OneHotCategoricalEncoder()),
('min max scaler',MinMaxScaler())



])
```

[70]:
```
time.fit(X_train,y_train)
```

[70]:
```
Pipeline(memory=None,
         steps=[('one hot encoding',
                 OneHotCategoricalEncoder(drop_last=False, top_categories=None,
                                          variables=['Education Level',
                                                     'Age Range', 'Gender'])),
                ('min max scaler',
                 MinMaxScaler(copy=True, feature_range=(0, 1)))],
         verbose=False)
```

[71]:
```
X_test=time.transform (X_test)
X_train=time.transform(X_train)
```

4.1 Logistic Regression

[81]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
clf = LogisticRegression(random_state=42)
clf.fit(X_train, y_train)

cv_scores = cross_val_score(clf, X_train, y_train)

# Mean Cross validation Score
print("Mean Cross-validation scores: {}".format(cv_scores.mean()))
print()



# Check test data set performance
print("Logistic  Train Performance: ", clf.score(X_train,y_train))
print("Logistic  Test Performance: ", clf.score(X_test,y_test))
```

```
Mean Cross-validation scores: 0.931546311297368

Logistic  Train Performance:  0.9325506651191858
Logistic  Test Performance:  0.9300072909071971
```

[82]:
```
from sklearn.metrics import classification_report

pred_test = clf.predict(X_test)
pred_train=clf.predict(X_train)
print(classification_report(y_test,pred))
```

```
                    precision    recall  f1-score   support

          Employed       1.00      0.97      0.98     12277
Not in labor force       0.82      0.99      0.90      5927
        Unemployed       0.72      0.11      0.18       998

          accuracy                           0.93     19202
         macro avg       0.85      0.69      0.69     19202
      weighted avg       0.93      0.93      0.92     19202
```

[93]:
```
from sklearn.metrics import r2_score, mean_squared_error
train_Rsquare = clf.score(X_train,y_train)
test_Rsquare = clf.score(X_test,y_test)
#train_MSE = mean_squared_error(y_train, pred_train)
#test_MSE = mean_squared_error(y_test, pred_test)
output = output.append(pd.Series({'model':'Logistic Regression','train_Rsquare':
↪train_Rsquare, 'test_Rsquare':test_Rsquare,'CV_Score':cv_scores.
↪mean()}),ignore_index=True )
output
```

[93]:
```
                 model  train_Rsquare  test_Rsquare  \
0  Logistic Regression       0.932551      0.930007
1  Logistic Regression       0.932551      0.930007
2  Logistic Regression       0.932551      0.930007

                                          CV_Score
0  [0.9311460774467135, 0.9304765093181565, 0.933...
1                                         0.931546
2                                         0.931546
```

4.2 KNN

[ ]:
```
# KNN

from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()

# define a list of parameters
```

```
param_knn = {'n_neighbors': range(1,10)}

#apply grid search
grid_knn = GridSearchCV(knn, param_knn, cv=5,verbose=True,␣
↪return_train_score=True,n_jobs=-1)
grid_knn.fit(X_train, y_train)

# Mean Cross Validation Score
print("Best Mean Cross-validation score: {:.2f}".format(grid_knn.best_score_))
print()

#find best parameters
print('KNN parameters: ', grid_knn.best_params_)

# Check train data set performance
print("KNN Train Performance: ", grid_knn.score(X_train,y_train))


# Check test data set performance
print("KNN Test Performance: ", grid_knn.score(X_test,y_test))
```

[ ]:
```
pred_test = grid_knn.predict(X_test)
pred_train=grid_knn.predict(X_train)
print(classification_report(y_test,pred_test))
```

[171]:
```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_dtree2 = BaggingClassifier(DecisionTreeClassifier(max_depth= 8,␣
↪max_leaf_nodes=5, min_samples_split= 3, splitter= 'random'), bootstrap=True,␣
↪random_state=0, oob_score=True)

bag_dtree2_param = {
                  'max_samples':  [0.8,1],
                  'n_estimators': [10,25,100]}
bag_dtree2_grid = GridSearchCV(bag_dtree2, bag_dtree2_param,cv=5,␣
↪n_jobs=-1,verbose=True,return_train_score=True, )
bag_dtree2_grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  1.3min finished
```

[171]:
```
GridSearchCV(cv=5, error_score=nan,
estimator=BaggingClassifier(base_estimator=DecisionTreeClassifier(ccp_alpha=0.0,
class_weight=None,
```

```
criterion='gini',
max_depth=8,
max_features=None,
max_leaf_nodes=5,
min_impurity_decrease=0.0,
min_impurity_split=None,
min_samples_leaf=1,
min_samples_split=3,
min_weight_fraction_leaf=0.0,
presort='deprecated',
random_state=None,
splitter='random'),
                                         bootstrap=True,
                                         bootstrap_features=False,
                                         max_features=1.0, max_samples=1.0,
                                         n_estimators=10, n_jobs=None,
                                         oob_score=True, random_state=0,
                                         verbose=0, warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_samples': [0.8, 1],
                         'n_estimators': [10, 25, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=True)
```

[172]:
```
print(f'Best Mean Cross Validation Score is {bag_dtree2_grid.best_score_}')
print(f'Best Mean Cross Validation Score is {bag_dtree2_grid.best_params_}')
print(f'Train score is {bag_dtree2_grid.score(X_train,y_train)}')
print(f'Test score is {bag_dtree2_grid.score(X_test,y_test)}')
y_pred = bag_dtree2_grid.predict(X_test)
print('Accuracy Score:',accuracy_score(y_test, y_pred))
```

```
Best Mean Cross Validation Score is 0.9343138695258821
Best Mean Cross Validation Score is {'max_samples': 0.8, 'n_estimators': 10}
Train score is 0.9331086510133024
Test score is 0.931048849078221
```

```
      ␣
↪---------------------------------------------------------------------------

      NameError                                 Traceback (most recent call␣
↪last)

      <ipython-input-172-e5ea436c66c0> in <module>
        4 print(f'Test score is {bag_dtree2_grid.score(X_test,y_test)}')
        5 y_pred = bag_dtree2_grid.predict(X_test)
  ----> 6 print('Accuracy Score:',accuracy_score(y_test, y_pred))
```

```
NameError: name 'accuracy_score' is not defined
```

[174]:
```python
pred_test = bag_dtree2_grid.predict(X_test)
pred_train=bag_dtree2_grid.predict(X_train)
print(classification_report(y_test,pred_test))
```

```
                   precision    recall  f1-score   support

          Employed       1.00      0.97      0.99     12277
Not in labor force       0.82      1.00      0.90      5927
        Unemployed       0.00      0.00      0.00       998

          accuracy                           0.93     19202
         macro avg       0.61      0.66      0.63     19202
      weighted avg       0.89      0.93      0.91     19202
```

4.3 Random Forest

[179]:
```python
from sklearn.ensemble import RandomForestClassifier
rfc =RandomForestClassifier(random_state=42)
rfc_param = {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'log2'],
    'max_depth' : [2,4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}

rfc_grid = GridSearchCV(rfc, rfc_param,cv=5, verbose=True,␣
 ↪return_train_score=True,n_jobs=-1 )
rfc_grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:  8.3min
[Parallel(n_jobs=-1)]: Done 240 out of 240 | elapsed: 11.2min finished
```

[179]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
```

```
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=None,
                                              oob_score=False, random_state=42,
                                              verbose=0, warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 4, 5, 6, 7, 8],
                         'max_features': ['auto', 'log2'],
                         'n_estimators': [200, 500]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=True)
```

[180]:
```
print(f'Best Mean Cross Validation Score is {rfc_grid.best_score_}')
print(f'Best Mean Cross Validation Score is {rfc_grid.best_params_}')
print(f'Train score is {rfc_grid.score(X_train,y_train)}')
print(f'Test score is {rfc_grid.score(X_test,y_test)}')
```

```
Best Mean Cross Validation Score is 0.9458084337685527
Best Mean Cross Validation Score is {'criterion': 'entropy', 'max_depth': 8,
'max_features': 'auto', 'n_estimators': 200}
Train score is 0.947616284260334
Test score is 0.9441724820331215
```

[181]:
```
pred_test = rfc_grid.predict(X_test)
pred_train=rfc_grid.predict(X_train)
print(classification_report(y_test,pred_test))
```

```
                    precision    recall  f1-score   support

          Employed       1.00      0.98      0.99     12277
Not in labor force       0.85      1.00      0.92      5927
        Unemployed       0.88      0.14      0.25       998

          accuracy                           0.94     19202
         macro avg       0.91      0.71      0.72     19202
      weighted avg       0.95      0.94      0.93     19202
```

4.4 Ada-Boost

[173]:
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
adc_dtree␣
↪=AdaBoostClassifier(base_estimator=DecisionTreeClassifier(),random_state=42)
adc_dtree_param = {
              'base_estimator__criterion' : ["gini", "entropy"],
              'base_estimator__splitter' :   ["best", "random"],
              'base_estimator__max_depth' : [2,4,6],
              'n_estimators' : [100,150],
              'learning_rate' : [0.5,0.2,1],
             }
adc_dtree_grid = GridSearchCV(adc_dtree, adc_dtree_param,cv=5, verbose=True,␣
↪return_train_score=True,n_jobs=-1 )
adc_dtree_grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  4.1min
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed: 29.2min
[Parallel(n_jobs=-1)]: Done 360 out of 360 | elapsed: 56.7min finished
```

[173]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=AdaBoostClassifier(algorithm='SAMME.R',
base_estimator=DecisionTreeClassifier(ccp_alpha=0.0,
class_weight=None,
criterion='gini',
max_depth=None,
max_features=None,
max_leaf_nodes=None,
min_impurity_decrease=0.0,
min_impurity_split=None,
min_samples_leaf=1,
min_samples_split=2,
min_weight_fraction_leaf=0.0,
presort='...
                                          learning_rate=1.0, n_estimators=50,
                                          random_state=42),
             iid='deprecated', n_jobs=-1,
             param_grid={'base_estimator__criterion': ['gini', 'entropy'],
                         'base_estimator__max_depth': [2, 4, 6],
                         'base_estimator__splitter': ['best', 'random'],
                         'learning_rate': [0.5, 0.2, 1],
                         'n_estimators': [100, 150]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=True)
```

[177]:
```
print(f'Best Mean Cross Validation Score is {adc_dtree_grid.best_score_}')
print(f'Best Mean Cross Validation Score is {adc_dtree_grid.best_params_}')
print(f'Train score is {adc_dtree_grid.score(X_train,y_train)}')
print(f'Test score is {adc_dtree_grid.score(X_test,y_test)}')
```

```
Best Mean Cross Validation Score is 0.9467235176837725
Best Mean Cross Validation Score is {'base_estimator__criterion': 'gini',
'base_estimator__max_depth': 2, 'base_estimator__splitter': 'best',
'learning_rate': 0.2, 'n_estimators': 100}
Train score is 0.946812784572806
Test score is 0.9450057285699406
```

[178]:
```
pred_test = adc_dtree_grid.predict(X_test)
pred_train=adc_dtree_grid.predict(X_train)
print(classification_report(y_test,pred_test))
```

```
                    precision    recall  f1-score   support

          Employed       1.00      0.98      0.99     12277
Not in labor force       0.85      1.00      0.92      5927
        Unemployed       0.88      0.16      0.27       998

          accuracy                           0.95     19202
         macro avg       0.91      0.71      0.73     19202
      weighted avg       0.95      0.95      0.93     19202
```

4.5 XG Boost

[182]:
```
from xgboost import XGBClassifier
from  xgboost import XGBClassifier
xgbc= XGBClassifier(random_state=42,early_stopping_rounds=2,objective= 'binary:
 ↪logistic')
xgbc_param = {
             'max_depth' : [2,4,6],
             'n_estimators' : [50,100,150],
             'learning_rate' : [0.1,0.5,0.6,0.8],
              'min_child_weight' : [1,3,5,7],
               #'subsample':[0.6,0.7,0.8,0.9,1]
            }
xgbc_grid = GridSearchCV(xgbc, xgbc_param,cv=5, verbose=True,␣
 ↪return_train_score=True,n_jobs=-1 )
xgbc_grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 144 candidates, totalling 720 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  2.9min
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed: 24.0min
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed: 52.7min
[Parallel(n_jobs=-1)]: Done 720 out of 720 | elapsed: 91.8min finished
```

[182]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None,
                                     early_stopping_rounds=2, gamma=None,
                                     gpu_id=None, importance_type='gain',
                                     interaction_constraints=None,
                                     learning_rate=None, max_delta_step=None,
                                     max_depth=None, min_child_weight=None,
                                     missing=nan, monotone_...
                                     random_state=42, reg_alpha=None,
                                     reg_lambda=None, scale_pos_weight=None,
                                     subsample=None, tree_method=None,
                                     validate_parameters=False,
                                     verbosity=None),
             iid='deprecated', n_jobs=-1,
             param_grid={'learning_rate': [0.1, 0.5, 0.6, 0.8],
                         'max_depth': [2, 4, 6],
                         'min_child_weight': [1, 3, 5, 7],
                         'n_estimators': [50, 100, 150]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=True)
```

[183]:
```
print(f'Best Mean Cross Validation Score is {xgbc_grid.best_score_}')
print(f'Best Parameters are {xgbc_grid.best_params_}')
print(f'Train score is {xgbc_grid.score(X_train,y_train)}')
print(f'Test score is {xgbc_grid.score(X_test,y_test)}')
```

```
Best Mean Cross Validation Score is 0.9470583117118625
Best Parameters are {'learning_rate': 0.1, 'max_depth': 4, 'min_child_weight':
1, 'n_estimators': 100}
Train score is 0.9477278814391572
Test score is 0.9452661181126966
```

[184]:
```
pred_test = xgbc_grid.predict(X_test)
pred_train=xgbc_grid.predict(X_train)
print(classification_report(y_test,pred_test))
```

```
              precision    recall  f1-score   support
```

Employed	1.00	0.98	0.99	12277
Not in labor force	0.85	1.00	0.92	5927
Unemployed	0.85	0.17	0.29	998
accuracy			0.95	19202
macro avg	0.90	0.72	0.73	19202
weighted avg	0.95	0.95	0.93	19202