Understanding_Happiness

May 6, 2020

1 Understanding Happiness

Christoph Uhl, 12/7/2019.

All code is my own.

But comments like `this`, are my professor's words. They guided me through the first part. Normal text I wrote myself.

`In this assignment I will explore happiness in the US using the General Social Survey, a database of survey results conducted by NORC at the University of Chicago from 1972 to present. The main research question here is: what makes people happy? Or, more specifically, what activities or life choices are most clearly correlated with self-reported happiness. And, over a long period of time, how have these trends changed.`

That is the introduction provided to me from my professor. But in my own words:

In this notebook I first analyze data and attempt to find the answer to questions provided by my professor. Then, I pose my own hypotheses and test them, following only my intuition.

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

1.1 Part 1: Training Wheels

`To get you started, well walk through some basic data processing and analysis.`

`Your task is to perform some statistical analyses to look at correlations between happiness and other variables, and then to fit a model using some of these variables.`

`First, lets read in the first 1000 rows of the data to have a look at it:`

[2]:
```
shortDf = pd.read_csv("gss.csv.bz2", nrows = 1000)
shortDf.head()
```

[2]:

	YEAR	ID	WRKSTAT	HRS1	HRS2	EVWORK	OCC	PRESTIGE	WRKSLF	WRKGOVT	\
0	1972	1	1	-1	-1	0	205	50	2	0	
1	1972	2	5	-1	-1	1	441	45	2	0	
2	1972	3	2	-1	-1	0	270	44	2	0	
3	1972	4	1	-1	-1	0	1	57	2	0	
4	1972	5	7	-1	-1	1	385	40	2	0	

```
   ...   NEISAFE  RLOOKS  RGROOMED  RWEIGHT  RHLTHEND  WTSS  WTSSNR  WTSSALL  \
0  ...         0       0         0        0         0     1       1   0.4446
1  ...         0       0         0        0         0     1       1   0.8893
2  ...         0       0         0        0         0     1       1   0.8893
3  ...         0       0         0        0         0     1       1   0.8893
4  ...         0       0         0        0         0     1       1   0.8893

   VSTRAT  VPSU
0      -1    -1
1      -1    -1
2      -1    -1
3      -1    -1
4      -1    -1

[5 rows x 6108 columns]
```

There are a few questions (columns) that relate to the respondants happiness. Lets take a look at those:

[3]:
```
print(shortDf["HAPPY7"].describe())
print(shortDf["HAPPY"].describe())
print(shortDf["HAPUNHAP"].describe())
```

```
count    1000.0
mean        0.0
std         0.0
min         0.0
25%         0.0
50%         0.0
75%         0.0
max         0.0
Name: HAPPY7, dtype: float64
count    1000.00000
mean        1.86600
std         0.79415
min         1.00000
25%         1.00000
50%         2.00000
75%         2.00000
max         9.00000
Name: HAPPY, dtype: float64
count    1000.0
mean        0.0
std         0.0
min         0.0
25%         0.0
50%         0.0
75%         0.0
max         0.0
```

```
Name: HAPUNHAP, dtype: float64
```

According to the data and the code book, HAPPY is the only one with good coverage over the duration of data collection, so lets focus on that one.

We need to make sure that we look at the data in its categorical meaning. Therefore, we will simmply replace all of the values in the Happy column based on the dictionary 'happyCategories'

[4]:
```
#lets tell the program that this data is categorical.
happyCategories = {1:"Very happy",2:"Pretty happy",3:"Not too happy",8:"Don't know",9:"No answer",0:"Not applicable"}
shortDf["HAPPY"] = shortDf["HAPPY"].replace(happyCategories) #map(lambda x: myDict[x], shortDf["HAPPY"])
shortDf["HAPPY"].describe()
```

[4]:
```
count             1000
unique               4
top       Pretty happy
freq               544
Name: HAPPY, dtype: object
```

Looks like most people are 'pretty happy'! Let's hypothesize that happy people are also healthy people. Let's check it out!

Converting the numeric values to corresponding categorical values just like before, but with different names.

[5]:
```
healthCategories = {1:"Excellent", 2:"Good", 3:"Fair", 4:"Poor", 8:"Dont know", 9:"No answer", 0:"Not applicable"}
shortDf["HEALTH"] = shortDf["HEALTH"].replace(healthCategories) #map(lambda x: myDict[x], shortDf["HAPPY"])
shortDf["HEALTH"].describe()
```

[5]:
```
count     1000
unique       5
top       Good
freq       450
Name: HEALTH, dtype: object
```

Really what we want to do is get a feel for how happy and health columns are related, so we can use a contingency table. However, there datapoints in the columns that ar 'no answer' and 'not applicable'. So we should probably not include them.

[6]:
```
cleanDf = shortDf.loc[(shortDf["HAPPY"] != "Not applicable") &(shortDf["HAPPY"] != "Dont know") & (shortDf["HAPPY"] != "No answer")]
cleanDf = cleanDf.loc[(cleanDf["HEALTH"] != "Not applicable") &(cleanDf["HEALTH"] != "Dont know") & (cleanDf["HEALTH"] != "No answer")]
# cleanDf = shortDf.loc[shortDf["HAPPY"] != 0]
# cleanDf = cleanDf[cleanDf["HAPPY"] != 9]
```

[7]:
```
print(cleanDf["HAPPY"].describe())
print(cleanDf["HEALTH"].describe())
```

```
count              995
unique               3
```

```
top       Pretty happy
freq               543
Name: HAPPY, dtype: object
count      995
unique       4
top       Good
freq       450
Name: HEALTH, dtype: object
```

[8]:
```
contingency_table = pd.crosstab(cleanDf["HAPPY"], cleanDf["HEALTH"],␣
↪margins=True)
contingency_table = contingency_table[["Poor", "Fair", "Good", "Excellent",␣
↪"All"]]
contingency_table
```

[8]:

HEALTH	Poor	Fair	Good	Excellent	All
HAPPY					
Not too happy	19	38	59	29	145
Pretty happy	24	113	265	141	543
Very happy	6	51	126	124	307
All	49	202	450	294	995

1.1.1 ————————- Q1 ————————-

Q1: Take a look at this contingency table. From a first look, does it appear that theres any relationship between the variables? Are healthy people more likely to be happy or unhappy? How about unhealthy people?

Looking at this contingency table, it seems like that there is a visible relationship between the Health and Happy variables. The best value for Health - Excellent - has a low count with the worst Happy variable ('Not too happy'), and a high count with the medium and highest Happy variables.

Health's 'Good' category has really high count with Happy's 'Pretty happy' and a high count with the 'Very happy' category. Albeit 'Good' health also has the highest count with the 'Not too happy' out of any health category.

Health's 'Fair' category has a similar low count for Happy's 'Not too happy', and 'Very happy' cateogries and a high count . However, since there are less than half as many 'Not too happy entries' as there are 'Very happy' entries, Health's fair category has a higher proportion of counts to Happy's 'Not too happy' category. I.e. people that indicated that they were 'not too happy' are more likely to choose that their health is 'Fair' than those that indicated that they were 'very happy'.

Health's 'Poor' category has a relatively high count for 'not too happy' and 'pretty happy', but here again, since there are 4 times more counts for the 'pretty happy' category than the'not too happy' category, there is a high relation between poor health and being 'not too happy'. There also seems to be a strong indication that people with poor health do not indicate that they are 'very happy'.

To sum it up, it looks like healthy people are more likely to be happy than unhappy. And unhealthy people are also more likely to indicate that they are unhappy rather than being happy.

Using a Chi-squared test, we can measure how likely it is that these data are independent (i.e., that there is no correlation or dependency between them).

1.1.2 ————————- Q2 ————————-

Q2: Try performing a chi-squared test on these two factors. Are they independent or does it appear that happiness and health are interrelated? Why?

Using a chi-squared test to measure the likelihood of these data being independent:

[10]:
```
from scipy import stats

usable = contingency_table.iloc[:-1,:-1]
obs = usable.values.tolist()
indices = list(usable.index)
cols = list(usable.columns)

chi2_stat, p_val, dof, expVals = stats.chi2_contingency(obs)
print("Chi-squared test returns a p-value of:", p_val)
```

```
Chi-squared test returns a p-value of: 1.4625840402568319e-09
```

The returned p-value of a chi-squared test is $p = 1.46 \cdot 10^{-9}$. Which tells us that we have statistically significant evidence to reject a null-hypthesis of the two variables 'HEALTH' and 'HAPPY' being independent, thus showing that they are very likely dependent variables.

The variables "HEALTH" and "HAPPY" being related makes a lot of sense, since healthy people do not have as many physical restraints from doing the things that makes them happy. Furthermore, those that are healthy likely have more energy available to spend on whatever they feel like, thus giving them more feelings of abundance.

Next, let's look at the numeric variable 'AGE'.

[11]:
```
ageList = cleanDf["AGE"].tolist()
ageList.sort(reverse=True)
print(ageList[:10])
print(len(ageList))
```

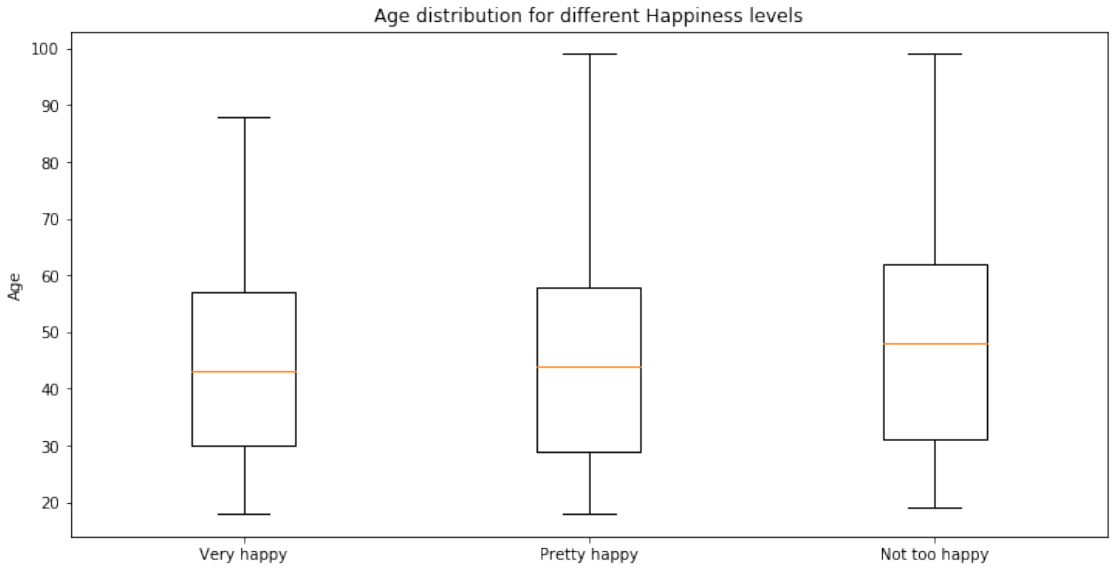
```
[99, 99, 88, 88, 85, 85, 84, 84, 84, 83]
995
```

As you can see, age 99 is likely the highest that this survey allowed for, and likely these ages were indicated by people who may not have take the survey very seriously. Nonetheless, for now, let's not worry about this, as there are only 2 such instances out of 995. It will add a bit of noise, but that is okay.

Let's make a boxplot with whiskers:

[12]:
```
boxes = [cleanDf[cleanDf["HAPPY"] == cat]["AGE"] for cat in happyCategories.
 ↪values()][:3]

fig,ax = plt.subplots(figsize=(12,6))
ax.boxplot(boxes)
ax.set_xticklabels(list(happyCategories.values())[:3])
```

```
ax.set_ylabel("Age")
ax.set_title("Age distribution for different Happiness levels")

plt.show()
```



The 99 values actually make the boxplot look different than it would be without. So let's try again but without age variables of 99. Please note that the following example drops, entirely, the rows that include age=99.

[13]:
```
altDf = cleanDf[cleanDf["AGE"] != 99]
alt_boxes = [altDf[altDf["HAPPY"] == cat]["AGE"] for cat in happyCategories.
 ↪values()][:3]

fig,ax = plt.subplots(figsize=(12,6))
ax.boxplot(alt_boxes)
ax.set_xticklabels(list(happyCategories.values())[:3])
ax.set_ylabel("Age")
ax.set_title("Age distribution for different Happiness levels")

plt.show()
```

Surprisingly different! To the human eye at least... Although, even with this change, all 3 boxplots look about the same, so I would predict that we won't find much dependency. We might find a slight correlation, as the median age of those that chose 'Not too happy' is a little higher than the other two.

So let's not count on the human eye and use a statistical test to better understand their relationships.

`We can use an Analysis of Variance (ANOVA) to test whether theres a significant difference in the mean age between each happiness group.`

The idea is that we decide if the means of the different distributions are approximately the same, or if they are different. Since we have more than 2 variables, let use ANOVA.

$H_0 : \mu_0 = \mu_1 = \mu_2$

$H_1 : \mu_i != \mu_j$, for at least one pair i, j

Analysis of Variance (ANOVA):

[15]:
```
fstat, pval = stats.f_oneway(boxes[0], boxes[1], boxes[2])
alt_fstat, alt_pval = stats.f_oneway(alt_boxes[0], alt_boxes[1], alt_boxes[2])
print("Without dropping rows containing age=99, we have a p-value = {}".
 ↪format(pval))
print("When dropping rows containing age=99, we have a p-value = {}".
 ↪format(alt_pval))
```

```
Without dropping rows containing age=99, we have a p-value = 0.04049654686481286
When dropping rows containing age=99, we have a p-value = 0.06180447480333926
```

1.1.3 ————————- Q3 ————————-

`Q3: Does age explain happiness according to the ANOVA? Why or why not?`

The p-value of ANOVA is 0.0418. This means that with an alpha of 0.05, there is statistically significant evidence to reject the null hypothesis of the mean of the three distributions being the same.

This means that according to ANOVA, age does explain happiness. They are dependent.

When looking at the closeness of the p-value to a relatively forgiving alpha, I am inclined to think that the relationship is really not that clear. Of course, following the statistical test I will assume that there really is a relationship between Age and Happiness, but preferably I would like to perform the test on more data.

Now load the full data instead of just the first 1,000 rows.

For further analysis, let's load in the full data.

[16]:
```
fullDf = pd.read_csv("gss.csv.bz2")
```

```
/anaconda3/lib/python3.7/site-packages/IPython/core/interactiveshell.py:2785:
DtypeWarning: Columns (19,20,21,25,26,49,50,51,55,56,67,68,69,73,74,80,81,82,86,
87,1572,1576,1579,1580,1635,1639,1642,1643,1690,1694,1697,1698,2365,5755) have
mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Now lets try to look at long term trends grouping respondants into categories by year.

And let's look at yearly averages of health and happiness.

[18]:
```
#so i want to group the data by year and then calculate the average of the␣
 ↪health and the average of the happy column
hhmeansDf = fullDf.groupby(["YEAR"])["HEALTH", "HAPPY"].mean()
hhmeansDf.head()
```

[18]:

YEAR	HEALTH	HAPPY
1972	2.010539	1.893366
1973	2.051197	1.791223
1974	2.025606	1.771563
1975	2.024161	1.826174
1976	2.027352	1.784523

Let's also calculate the percentage of people who are very happy or in excellent health. This approach is better for this data, since it's categorical.

[19]:
```
print(fullDf["HEALTH"].unique())
print(fullDf["HAPPY"].unique())
```

```
[2 3 1 4 9 8 0]
[3 2 1 9 8 0]
```

Category '1' for the health and happy variables both indicate the highest option ("Excellent" and "Very Happy" respectively).

Also, let's drop the rows with 0. If we don't there will be some years that have no answers with 1s, simply because it wasn't a question during those years.

[20]:
```
#dropping 0s
fullDf = fullDf[fullDf["HEALTH"] != 0]
fullDf = fullDf[fullDf["HAPPY"] != 0]
```

[21]:
```
#so we need to calculate the number of rows with a 1 in EITHER the health OR␣
 ↪the happy column, and divide that by total number of rows.
both_1s = entries_with_1s = fullDf.loc[(fullDf["HEALTH"] == 1) &␣
 ↪(fullDf["HAPPY"] == 1)]
entries_with_1s = fullDf.loc[(fullDf["HEALTH"] == 1) | (fullDf["HAPPY"] == 1)]
health_with_1s = fullDf.loc[(fullDf["HEALTH"] == 1)]
happy_with_1s = fullDf.loc[(fullDf["HAPPY"] == 1)]

meta_1s = [both_1s, entries_with_1s, health_with_1s, happy_with_1s]
```

[22]:
```
print("BOTH HEALTH AND HAPPY = 1")
print("Number of rows with a 1 in both: {}, number of total rows: {}".
 ↪format(both_1s.shape[0], fullDf.shape[0]))
print("Percentage of people who are both very happy and in excellent health: {:.
 ↪1f}".format(100*both_1s.shape[0]/fullDf.shape[0]))
print()
print("EITHER HEALTH OR HAPPY = 1")
print("Number of rows with a 1 in either: {}, number of total rows: {}".
 ↪format(entries_with_1s.shape[0], fullDf.shape[0]))
print("Percentage of people who are very happy or in excellent health: {:.1f}".
 ↪format(100*entries_with_1s.shape[0]/fullDf.shape[0]))
print()
print("HEALTH = 1")
print("Number of rows with a 1 in health: {}, number of total rows: {}".
 ↪format(health_with_1s.shape[0], fullDf.shape[0]))
print("Percentage of people who are in excellent health: {:.1f}".
 ↪format(100*health_with_1s.shape[0]/fullDf.shape[0]))
print()
print("HAPPY = 1")
print("Number of rows with a 1 in happy: {}, number of total rows: {}".
 ↪format(happy_with_1s.shape[0], fullDf.shape[0]))
print("Percentage of people who are very happy: {:.1f}".
 ↪format(100*happy_with_1s.shape[0]/fullDf.shape[0]))
```

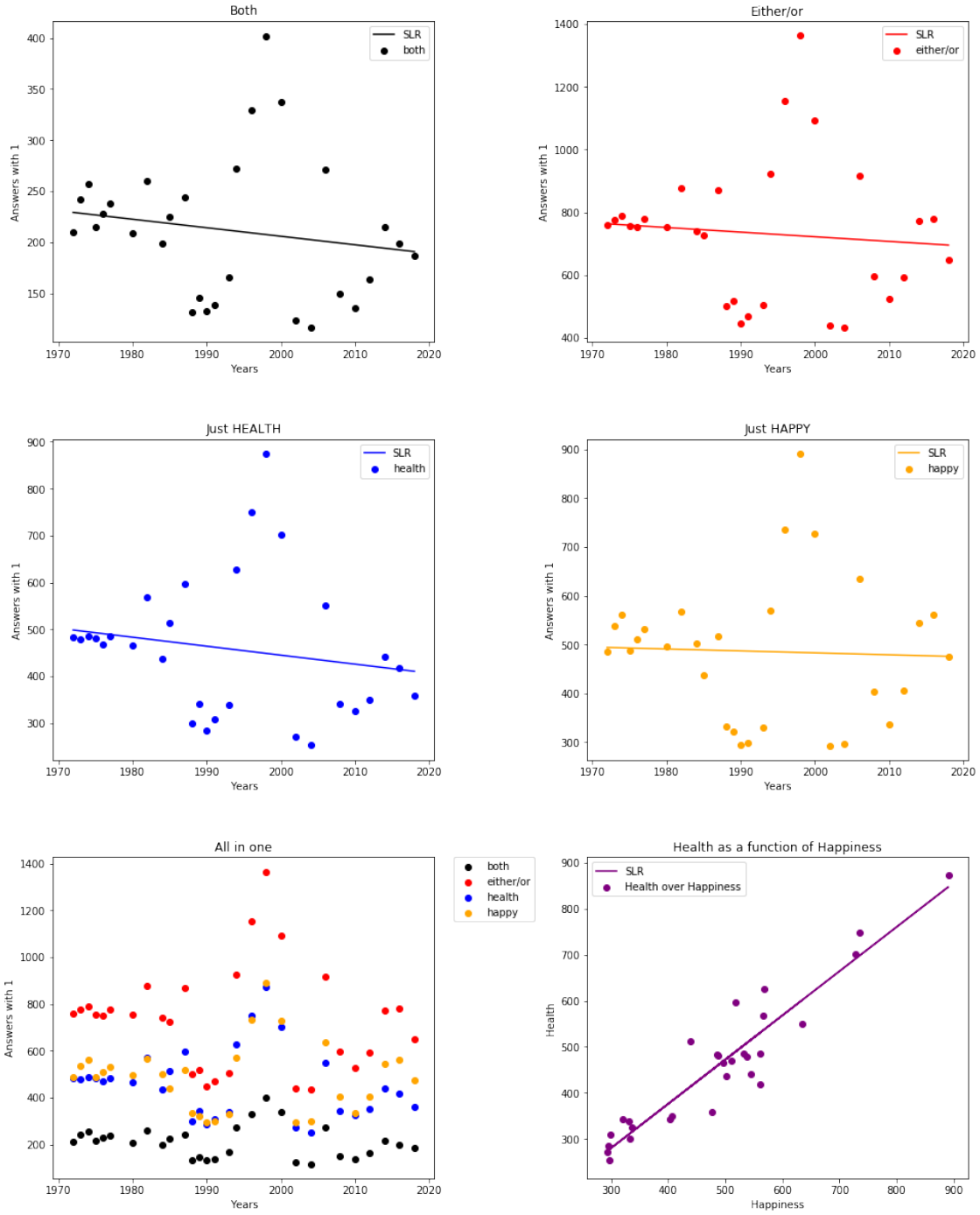
```
BOTH HEALTH AND HAPPY = 1
Number of rows with a 1 in both: 6146, number of total rows: 44810
Percentage of people who are both very happy and in excellent health: 13.7

EITHER HEALTH OR HAPPY = 1
Number of rows with a 1 in either: 21246, number of total rows: 44810
Percentage of people who are very happy or in excellent health: 47.4

HEALTH = 1
Number of rows with a 1 in health: 13302, number of total rows: 44810
Percentage of people who are in excellent health: 29.7

HAPPY = 1
```

```
Number of rows with a 1 in happy: 14090, number of total rows: 44810
Percentage of people who are very happy: 31.4
```

You can make a scatterplot of these data. You might want to plot a single variable over time, or plot health and happiness against each other.

[23]:
```
colors = ['black', 'red', 'blue', 'orange', 'purple']
data_names = ['both', 'either/or', 'health', 'happy']
lins = []
pearson_ccoef = []
xypairs = []
#TOP ROW
fig, ax = plt.subplots(3,2, figsize=(16,16))
for i,name in enumerate(data_names[:2]):
    x = meta_1s[i]["YEAR"].unique()
    y = meta_1s[i].groupby(["YEAR"])["ID"].count()
    xypairs.append((x,y))
    #Get a pearson correlation coefficient:
    pearson_ccoef.append(stats.pearsonr(x, y))

    #get SLR to help with visualization
    slope, intercept, r_value, p_value, std_err = stats.linregress(x,y)
    #store for cumulative plot
    lins.append((slope, intercept, r_value, p_value, std_err))
    #plot SLR
    ax[0][i].plot(x, intercept + slope*x, color = colors[i])
    #plot data
    ax[0][i].scatter(x, y, color = colors[i])
    ax[0][i].legend(["SLR", name])
    ax[0][i].set_xlabel("Years")
    ax[0][i].set_ylabel("Answers with 1")
    ax[0][i].set_title(name.capitalize())


#SECOND ROW
for i,name in enumerate(data_names[2:]):
    x = meta_1s[i+2]["YEAR"].unique()
    y = meta_1s[i+2].groupby(["YEAR"])["ID"].count()
    xypairs.append((x,y))
    #Get a pearson correlation coefficient:
    pearson_ccoef.append(stats.pearsonr(x, y))

    #get SLR to help with visualization
    slope, intercept, r_value, p_value, std_err = stats.linregress(x,y)
    #store for cumulative plot
    lins.append((slope, intercept, r_value, p_value, std_err))
    #plot SLR
    ax[1][i].plot(x, intercept + slope*x, color = colors[i+2])
```

```
    #plot data
    ax[1][i].scatter(x, y, color = colors[i+2])
    ax[1][i].legend(["SLR", name])
    ax[1][i].set_xlabel("Years")
    ax[1][i].set_ylabel("Answers with 1")
    ax[1][i].set_title("Just " + name.upper())

#ALL IN ONE
for i in range(4):
    x = xypairs[i][0]
    y = xypairs[i][1]
#     print(i)
#     ax[2][1].plot(x, lins[i][1] + lins[i][0]*x, color = colors[i])
    ax[2][0].scatter(x, y, color = colors[i])
ax[2][0].set_title("All in one")
ax[2][0].set_xlabel("Years")
ax[2][0].set_ylabel("Answers with 1")
#for the legend
# SLR_labels = []
labels = []
for name in data_names:
#     SLR_labels.append(name + " SLR")
    labels.append(name)
ax[2][0].legend(labels, bbox_to_anchor=(1.05, 1), loc='upper left',␣
 ↪borderaxespad=0.)


#Health vs Happiness
x = meta_1s[3].groupby(["YEAR"])["ID"].count() #happiness
y = meta_1s[2].groupby(["YEAR"])["ID"].count() #health
xypairs.append((x,y))
lins.append(stats.linregress(x, y))
ax[2][1].plot(x, lins[-1][1] + lins[-1][0]*x, color = colors[4])
ax[2][1].scatter(x, y, color = colors[4])
ax[2][1].set_title("Health as a function of Happiness")
ax[2][1].set_xlabel("Happiness")
ax[2][1].set_ylabel("Health")
ax[2][1].legend(["SLR", "Health over Happiness"])

plt.subplots_adjust(top = 0.99, bottom=0.01, hspace=.3, wspace=0.4)
plt.show()
```



1.1.4 ———————- Q4 ———————-

Q4: To understand how well these variables are correlated, you can use Pearsons correlation and/or fit a linear regression. If you fit a linear regression, spend some time researching the assumptions. Calculate the correlation coefficient and

p-value for each combination of variable (health and year, happiness and year, health and happiness). Are they correlated? How well? What does this mean?

Overall, there doesn't seem to be any significant change over the last 50 years. The SLRs aren't the best predictor for our variables, but we aren't trying to predict right now, mainly just understanding the general flow of the data, which is relatively horizontal. Let's look at the SLR p-value and r squared:

[24]:
```
for i, name in enumerate(data_names):
    print("For {}, we have p-value = {}, r-value = {}".format(name, lins[i][3],␣
 ↪lins[i][2]))
print("For {}, we have p-value = {}, r-value = {}".format("Happy over Health",␣
 ↪lins[4][3], lins[4][2]))
```

```
For both, we have p-value = 0.3798736238626914, r-value = -0.16933079902605425
For either/or, we have p-value = 0.632111773144608, r-value =
-0.09279232609694471
For health, we have p-value = 0.35210154847149255, r-value = -0.1792735609542711
For happy, we have p-value = 0.84459475833553, r-value = -0.038061005640476916
For Happy over Health, we have p-value = 5.812581081328852e-14, r-value =
0.9382193145992908
```

The assumptions that a simple linear regression makes are:

1) The data can be modeled by:
$$y_i = \alpha + \beta x_i + \epsilon_i$$

2) Each of the ϵ_i are independent.

3) $\epsilon_i \sim N(0, \sigma^2)$

Health and year, as well as happiness and year are not well correlated according to this linear regression. This means that neither health, nor happiness, as had a significant change during the last 50 years, according to the data. Happiness, even more so than health shows a near horizontal linear regression. However, I can't help to wonder, "If we were to normalize this data based on the number of responses for each year, what would this correlation linear regression look like? How would it change?".

For Health over Happiness, there does seem to be a very strong correlation. Based on the p-values of the linear regressions, there is significant evidence health and happiness change together. This COULD mean that a person's happiness influences a person's health. However, there is a potential confounding variable which would be the number of total responses to the survey, and furthermore it could also be a health that influences happiness. Simply put, as the count for people indicating they are "very happy" increases, so does the count of people indicating they are in "excellent" health. This means that people who are happy are also healthy, but it does not necessarily mean that as people become happier, they also become healthier.

1.1.5 ————————- Q5: Fitting an MLR model ————————-

Q5: Next, fit a regression model with HAPPY as the dependent variable, and YEAR and HEALTH as the independent variables (features). How good is this regression model? What does it mean?

Next, fit a regression model with HAPPY as the dependent variable, and YEAR and HEALTH as the independent variables (features). How good is this regression model? What does it mean?

Now, let's look at the data in a single model.

[25]:
```
import statsmodels.api as sm

# Collect the features in a 2D array
mlrDf = pd.DataFrame({"YEAR": meta_1s[2]["YEAR"].unique(), "HEALTH": meta_1s[2].
 ↪groupby(["YEAR"])["HEALTH"].count()})
X = mlrDf
# print(X.head())
# Add a constant to the array for the intecept
X = sm.add_constant(X)

# Collect the response data in an array
y = meta_1s[3].groupby(["YEAR"])["HAPPY"].count()

# Fit the ordinary least-squares (OLS) model
model = sm.OLS(y, X).fit()
```

[26]:
```
model.summary()
```

[26]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                   HAPPY   R-squared:                       0.898
Model:                             OLS   Adj. R-squared:                  0.890
Method:                  Least Squares   F-statistic:                     114.1
Date:                 Fri, 13 Dec 2019   Prob (F-statistic):           1.33e-13
Time:                         18:00:03   Log-Likelihood:                -152.16
No. Observations:                   29   AIC:                             310.3
Df Residuals:                       26   BIC:                             314.4
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -2737.1522   1329.085     -2.059      0.050   -5469.126      -5.178
YEAR           1.4010      0.664      2.109      0.045       0.036       2.766
HEALTH         0.9395      0.062     15.097      0.000       0.812       1.067
==============================================================================
Omnibus:                        0.900   Durbin-Watson:                   1.059
Prob(Omnibus):                  0.638   Jarque-Bera (JB):                0.838
Skew:                          -0.182   Prob(JB):                        0.658
Kurtosis:                       2.251   Cond. No.                     3.02e+05
==============================================================================

Warnings:
```

```
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 3.02e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

This regression model is really good, because it has a really high f-statistic. The f-statistic is a measure of how much better our model is than just using the mean. The larger it is, the better. You may also look at the adjusted R-squared value of 0.89. It's a really good model.

The t-statistic calculated for the "YEAR" variable is barely significant at the alpha = 5% level and for the "HEALTH" variable it is incredibly high at 15.097. There are not enough decimal places to show the p-value of it in the summary provided above. This means that as Happiness can be explained by Health and Year – The healthier a person is, the happier they are, and as time progresses into the future, people's average happiness also increases. The latter statement is rather questionable, as it should be, the model made the assumption that the data can be explained in a linear fashion; however it may just be that due to an increase in connectivity through the internet, people have become happier, but that this trend will not continue into the future. Of course it is also noteworthy that people, on average, may become healthier as time moves on, which would mean that time doesn't directly drive happiness.

If we look at the correlation between time and health, we don't see a statistically significant correlation. So I am wondering why the MLR model showed a p-value of 0.045. I think a reason for this may be the lack of data points. We only have 26 data points, so it may be a good idea to bootstrap some more data.

1.2 Part 2: On my own now

For the next part of the notebook, I will go out on my own:

1.2.1 ————————- Q6 ————————-

```
Q6: Define three hypotheses about additional variables that may be correlated
with happiness. Youll need to reference the code book. Why do you think these may
or may not be correlated with happiness?
```

1) Being self-employed increases happiness.
    - I think being an employer might make a person happier as they are providing an income to live on for people.
2) Income increases happiness.
    - I think that up to a certain point income should increase happiness, let's find out the truth.
3) People are happier in the summer.
    - I think sunshine and warmth can make people happier.

1.2.2 ————————- Q7 ————————-

```
Q7: Using methods like those in part 1, explore whether or not the data supports
your hypotheses.
```

1.2.3 1) Being self-employed increases happiness.

To test this hypothesis, let's first get a feel for the data by showing a contingency table.

[27]:
```
wrkHapDf = fullDf.loc[((fullDf["WRKSLF"] == 1) | (fullDf["WRKSLF"] == 2)) &␣
↪((fullDf["HAPPY"] == 1) | (fullDf["HAPPY"] == 2) | (fullDf["HAPPY"] ==␣
↪3))][["YEAR", "HAPPY", "WRKSLF"]]
```

[28]:
```
wrkCategories = {1:"Self-employed",2:"Someone else",8:"Don't know",9:"No␣
↪answer",0:"Not applicable"}
catWrkDf = wrkHapDf.copy()
catWrkDf["WRKSLF"] = wrkHapDf["WRKSLF"].replace(wrkCategories) #map(lambda x:␣
↪myDict[x], shortDf["HAPPY"])
catWrkDf["HAPPY"] = catWrkDf["HAPPY"].replace(happyCategories)
print(catWrkDf["WRKSLF"].describe())
print(catWrkDf["HAPPY"].describe())
print("How many rows there were before dropping them in the cell above: ",␣
↪fullDf.shape[0])
```

```
count             41810
unique                2
top       Someone else
freq              37007
Name: WRKSLF, dtype: object
count             41810
unique                3
top       Pretty happy
freq              23126
Name: HAPPY, dtype: object
How many rows there were before dropping them in the cell above:  44810
```

Let's take a look at the contingency table.

[29]:
```
contingency_table = pd.crosstab(catWrkDf["WRKSLF"], catWrkDf["HAPPY"],␣
↪margins=True)
print("The ratio in group size is approx. {}:{:.1f}".format(1,␣
↪catWrkDf[catWrkDf["WRKSLF"] == "Someone else"].shape[0]/
↪catWrkDf[catWrkDf["WRKSLF"] == "Self-employed"].shape[0]))
contingency_table
```

```
The ratio in group size is approx. 1:7.7
```

[29]:

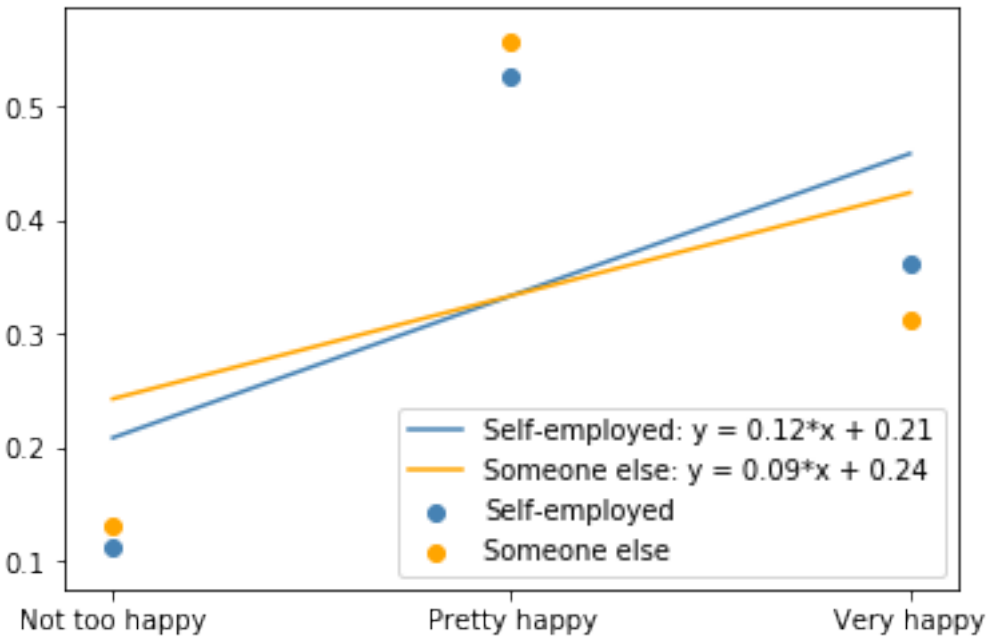
HAPPY	Not too happy	Pretty happy	Very happy	All
WRKSLF				
Self-employed	539	2525	1739	4803
Someone else	4849	20601	11557	37007
All	5388	23126	13296	41810

The count for each happy category should be approximately 7.7 times for the "Someone else" category than for the "Self-employed" category. However, for the "Not too happy" category, the count is 9 times higher for those working for "Someone else" as for those that are "Self-employed".

So there is a 1:9 ratio for the "Not too happy" category.
There is a 1:8.1 ratio for the pretty happy category.
And a 1:6.6 ratio for the very happy category.

Based on this, I would say that there is a correlation between being self-employed and happiness. However, the contingency table is really tough to read for this data, so let's normalize the data and plot it what an SLR.

[30]:
```
x = catWrkDf["HAPPY"].unique()
numeric_x = np.asarray([0.0, 1.0, 2.0])

selfEmp = catWrkDf[catWrkDf["WRKSLF"] == "Self-employed"]
someoneElse = catWrkDf[catWrkDf["WRKSLF"] == "Someone else"]
selfEmpCounts = [selfEmp[selfEmp["HAPPY"] == hapCat].shape[0] for hapCat in␣
 ↪list(x)]
elseCounts = [someoneElse[someoneElse["HAPPY"] == hapCat].shape[0] for hapCat␣
 ↪in list(x)]
total_self = sum(selfEmpCounts)
total_else = sum(elseCounts)
selfEmpCounts = [num/total_self for num in selfEmpCounts]
elseCounts = [num/total_else for num in elseCounts]

selfLin = stats.linregress(numeric_x, selfEmpCounts)
elseLin = stats.linregress(numeric_x, elseCounts)

plt.scatter(x, selfEmpCounts, color = "steelblue")
plt.plot(x, selfLin[1]+selfLin[0]*numeric_x, color = "steelblue")
plt.scatter(x, elseCounts, color = "orange")
plt.plot(x, elseLin[1]+elseLin[0]*numeric_x, color = "orange")
plt.legend(["Self-employed: y = {:.2f}*x + {:.2f}".format(selfLin[0],␣
 ↪selfLin[1]), "Someone else: y = {:.2f}*x + {:.2f}".format(elseLin[0],␣
 ↪elseLin[1]), "Self-employed","Someone else"])
plt.show()
```

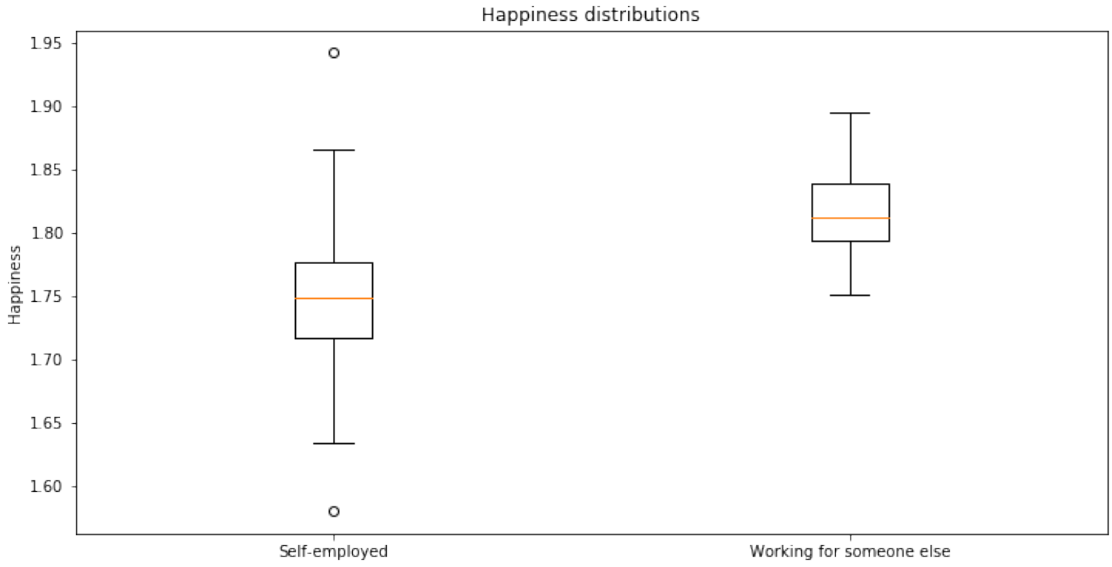


Since there are only 3 categories, it seems a bit weird to perform an SLR on the two, but just from plotting the data points, it is clear to see that those that are self-employed have a higher happiness, at least based on this data. We could construct a confidence interval for the slopes of the two SLRs and see how close they are, if they overlap, etc., but with only 3 datapoints, this seems not so helpful.

Another way to do this would be to calculate the mean happiness number (i.e. convert the categorical data back to its numerical form and average out over the two different employment groups) and then test if there is statistically significant evidence to reject a null hypothesis of the two true means being equal using ANOVA. Let's do that.

[50]:
```
# no real code that we need to implement to get back to the numerical values␣
 ↪since we made a copy for the categorical representation
#just use wrkHapDf
#also we need to a better idea of how the mean is spread out to do a boxplot.␣
 ↪So that's
elseMeans = wrkHapDf[wrkHapDf["WRKSLF"] == 2].groupby(["YEAR"])["HAPPY"].mean()
selfEmpMeans = wrkHapDf[wrkHapDf["WRKSLF"] == 1].groupby(["YEAR"])["HAPPY"].
 ↪mean()

fig,ax = plt.subplots(figsize=(12,6))
ax.boxplot([selfEmpMeans, elseMeans])
ax.set_xticklabels(["Self-employed", "Working for someone else"])
ax.set_ylabel("Happiness")
ax.set_title("Happiness distributions")

plt.show()
```

```
# selfEmpCounts = [selfEmp[selfEmp["HAPPY"] == hapCat].shape[0] for hapCat in␣
 ↪list(x)]
# elseCounts = [someoneElse[someoneElse["HAPPY"] == hapCat].shape[0] for hapCat␣
 ↪in list(x)]
# total_self = sum(selfEmpCounts)
# total_else = sum(elseCounts)
```



These results are rather surprising. Let's perform ANOVA to see if the means are the same.

[51]:
```
fstat, pval = stats.f_oneway(selfEmpMeans, elseMeans)
print(pval)
```

```
2.6425008606443248e-05
```

Based on this data, there is significant evidence that the means of happiness levels of self-employed and those who work for someone else are different based on a p-value of $ p = 2.64 * 10^{-5} $, which is far less than a reasonable alpha of 0.01.

I believe the latter representation to be more representative of the truth, since before we used an SLR to show the difference, but that only had 3 data points.

So overall, we can conclude that people who are self-employed less happy on average than those who are working for someone else.

1.2.4 2) Income increases happiness.

Let's begin this one by just plotting the numbers.

[52]:
```
incDf = fullDf.loc[(fullDf["RINCOME"] < 13) & (fullDf["RINCOME"] != 0)]
incDf = incDf.loc[(incDf["HAPPY"] < 4) & (incDf["HAPPY"] != 0)]
incDf = incDf[["YEAR", "HAPPY", "RINCOME"]]
incDf.head()
```

[52]:
```
      YEAR  HAPPY  RINCOME
3117  1974      1        2
3118  1974      1       10
3121  1974      2        7
3122  1974      2        8
3123  1974      1       11
```

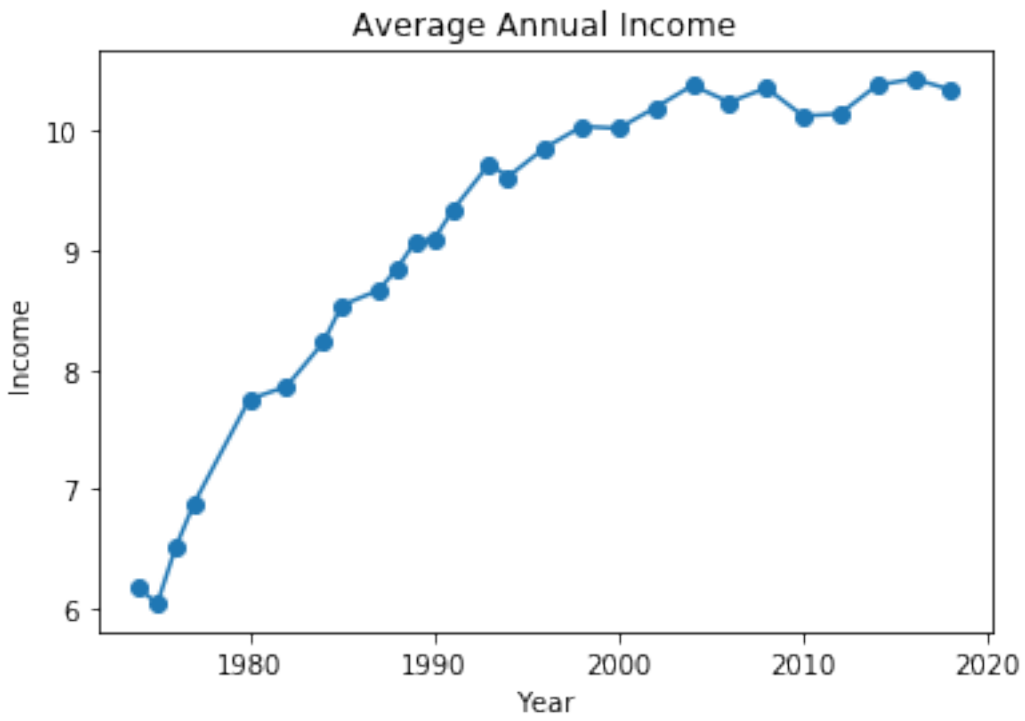
First, let's convert the income numbers into the written categories.

[53]:
```
incCategories = {1: "Under $1,000", 2: "$ 1,000 to 2,999", 3: "$ 3,000 to␣
↪3,999",
                 4: "$ 4,000 to 4,999", 5: "$ 5,000 to 5,999", 6: "$ 6,000 to␣
↪6,999",
                 7: "$ 7,000 to 7,999", 8: "$ 8,000 to 9,999", 9: "$10,000 to␣
↪14,999",
                 10: "$15,000 to 19,999", 11: "$20,000 to 24,999", 12: "$25,000␣
↪or over"}

catIncDf = incDf.copy()
catIncDf["RINCOME"] = incDf["RINCOME"].replace(incCategories)
catIncDf["HAPPY"] = catIncDf["HAPPY"].replace(happyCategories)
```

[54]:
```
print(catIncDf["RINCOME"].describe())
print(catIncDf["HAPPY"].describe())
```

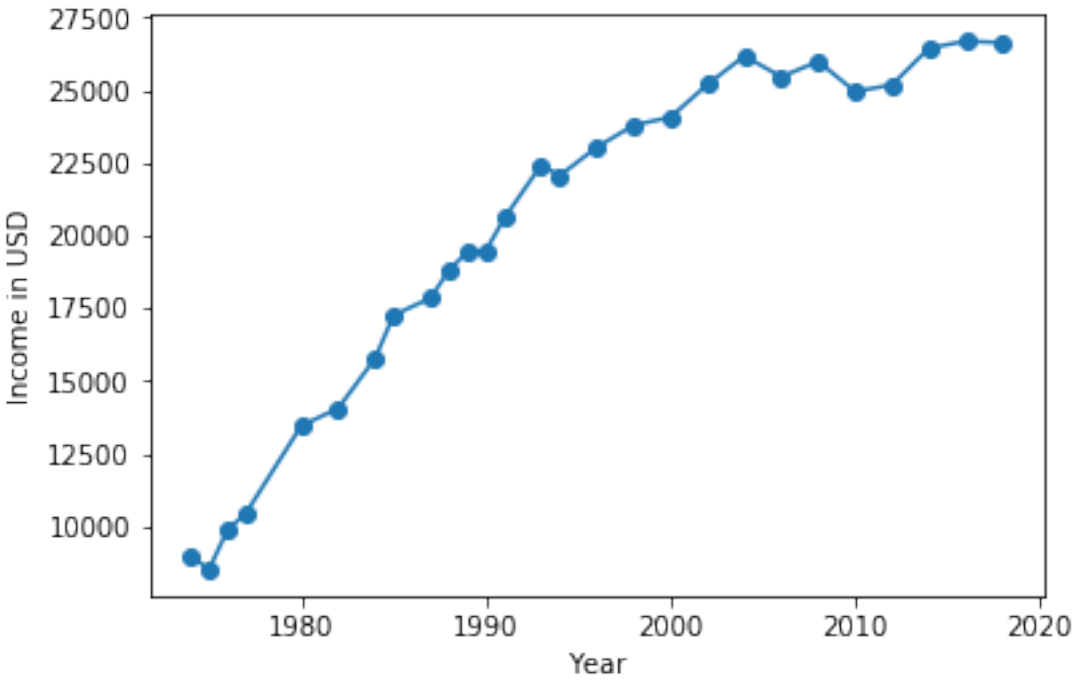
```
count                25508
unique                  12
top       $25,000 or over
freq                  9590
Name: RINCOME, dtype: object
count             25508
unique                3
top       Pretty happy
freq              14709
Name: HAPPY, dtype: object
```

[55]:
```
#let's plot income over year
x = incDf["YEAR"].unique()
inc = incDf.groupby(["YEAR"])["RINCOME"].mean()

plt.scatter(x,inc)
plt.plot(x, inc)
plt.xlabel("Year")
plt.ylabel("Income")
plt.title("Average Annual Income")
plt.show()
```



This is a nice little plot, but the values at the top are a little misleading. Let's see if we can make our y-axis show more exact monetary values, rather than categories.

[56]:
```
incCategories = {1: "Under $1,000", 2: "$ 1,000 to 2,999", 3: "$ 3,000 to␣
↪3,999",
                 4: "$ 4,000 to 4,999", 5: "$ 5,000 to 5,999", 6: "$ 6,000 to␣
↪6,999",
                 7: "$ 7,000 to 7,999", 8: "$ 8,000 to 9,999", 9: "$10,000 to␣
↪14,999",
                 10: "$15,000 to 19,999", 11: "$20,000 to 24,999", 12: "$25,000␣
↪or over"}

incCategories = {1: 1000, 2: 2000, 3: 3500, 4: 4500, 5: 5500, 6: 6500,
                 7: 7500, 8: 9000, 9: 12500, 10: 17500, 11: 22500, 12: 35000}
incValDf = incDf.copy()
incValDf["RINCOME"] = incDf["RINCOME"].replace(incCategories)
incVals = incValDf.groupby(["YEAR"])["RINCOME"].mean()
incValDf.head()

plt.scatter(x, incVals)
plt.plot(x, incVals)
plt.xlabel("Year")
plt.ylabel("Income in USD")
plt.show()
```



The problem with the above graph is that the highest *categorical* value possible represents 25,000USD or over. I have coded it to be just 30,000USD, but in reality this could be hundreds of thousands of dollars. There is just no way for me to really estimate that with a number, and therefore represent it well.

Nonetheless, it does show a clear trend of increase in income. If this is could be solely described by inflation, there would be an inflation rate of:

[57]:
```
listVals = list(incVals)
print("In 1974, avg. income = {}. In 2018, avg. income = {}".
 ↪format(listVals[0], listVals[-1]))
```

```
In 1974, avg. income = 8993.436754176611. In 2018, avg. income =
26623.006833712985
```

The average income in 1974 was 8,784.6 USD, which would be 44,743.88 USD in 2018 according to this website.

However, according to the model, there would only be 26,623.01 USD. The obvious fail of the model is the highest option for income category, which I estimated at 35,000. I think it would be best to fit a SLR based on the first 20 years of data and use that to predict the growth. But since we are actually trying to test the correlation between income and happiness, there is no need to go further.

What we have found by doing this analysis is that the more recent data can't be used well. So let's use data going up the year 2000 to determine the happiness - income correlation.

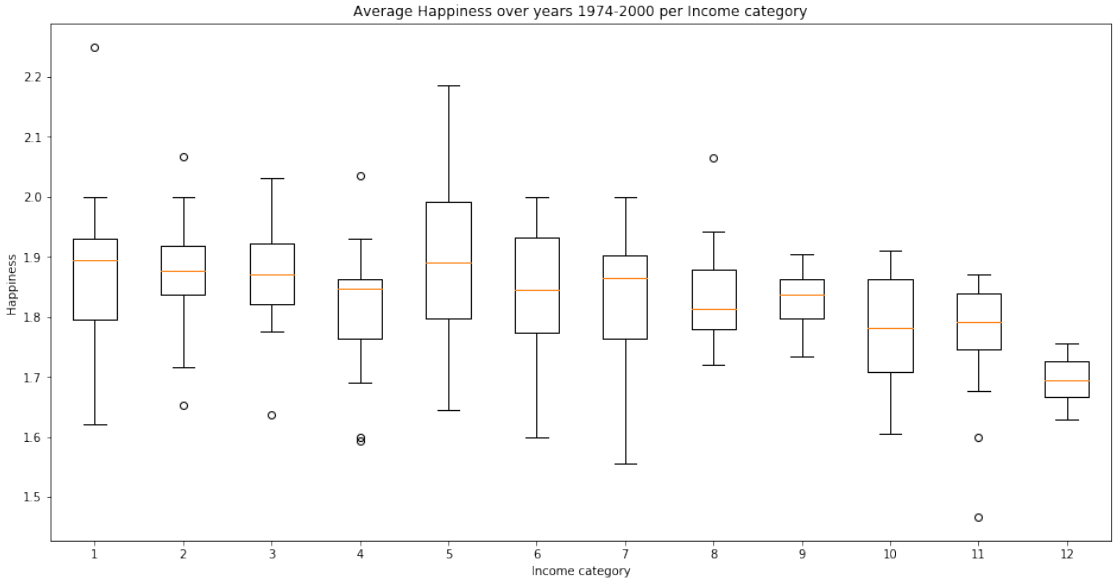
[58]:
```
incDf = incDf[incDf["YEAR"]<2001]
incDf.head()
```

[58]:

	YEAR	HAPPY	RINCOME
3117	1974	1	2
3118	1974	1	10
3121	1974	2	7
3122	1974	2	8
3123	1974	1	11

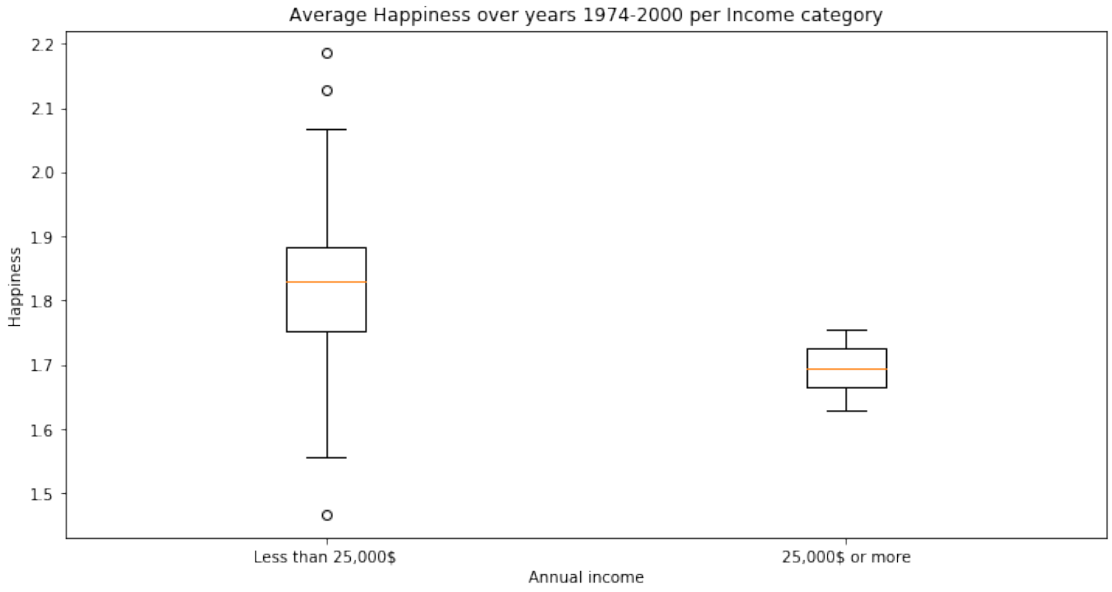
[59]:
```
#lets make a box plot for all of the different categories that could be chosen␣
 ↪in terms of income
boxDf = incDf.groupby(["RINCOME", "YEAR"]).mean()
```

[62]:
```
# for i in range(1, 13):
#     for j in range(1974, 2001):
#         boxDf.iloc[i,j]
boxes = []
for i in range(1,13):
    boxes.append(boxDf[(i, 1974):(i, 2000)]["HAPPY"])

fig, ax = plt.subplots(figsize=(16, 8))
ax.boxplot(boxes)
ax.set_xlabel("Income category")
ax.set_ylabel("Happiness")
ax.set_title("Average Happiness over years 1974-2000 per Income category")
plt.show()
```



So this gives us a nice overview of how the means of the data are distributed across years 1974-2000. Interestingly enough, those that chose the highest income level have the lowest mean Happiness level. Let's examine if the mean of the highest category is significantly different from the rest. This means that we combine all of the previous income categories into one category and test if its mean is different from the 12th category.

[63]:
```
#first combine the first 11
first11 = []
for i in range(1,12):
    first11 = first11 + list(boxes[i])

boxes_new = [first11, boxes[-1]]

fig, ax = plt.subplots(figsize=(12,6))
ax.boxplot(boxes_new)
ax.set_xticklabels(["Less than 25,000$", "25,000$ or more"])
ax.set_xlabel("Annual income")
ax.set_ylabel("Happiness")
ax.set_title("Average Happiness over years 1974-2000 per Income category")
plt.show()


print("Number of data points in first box: {}, num. dps in second box: {}".
 ↪format(len(first11), len(boxes[-1])))
```




```
Number of data points in first box: 198, num. dps in second box: 18
```

Let's use ANOVA:

[64]:
```
fstat, pval = stats.f_oneway(first11, boxes[-1])
print(pval)
```

```
3.258639653477285e-06
```

ANOVA yields a p-value of $p = 3.26 * 10^{-6}$, which is statistically significant at a reasonable alpha of 0.01.

This indicates that people with 25,000 dollar or more annual income on average are less happy than those that make less than that amount a year.

I believe there is good reason to be skeptical of these results, but for now let's move on to the next hypothesis.

1.2.5 3) People are happier in the summer

To test this, let's create our trusty boxplot and run an ANOVA on it.

[65]:
```
dateDf = fullDf.loc[(fullDf["DATEINTV"] != 9999) & (fullDf["DATEINTV"] != 0)]
dateDf = dateDf.loc[(dateDf["HAPPY"] < 4) & (dateDf["HAPPY"] != 0)]
dateDf = dateDf[["YEAR", "HAPPY", "DATEINTV"]]
dateDf.head()
```

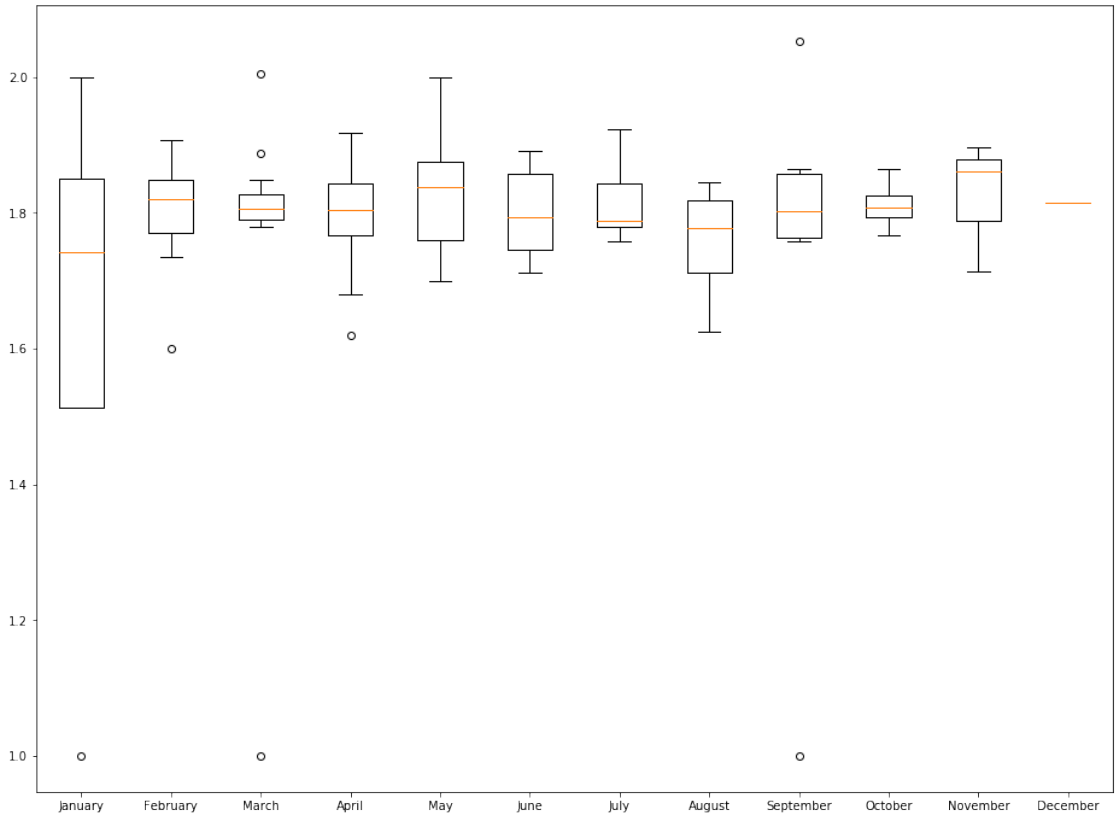
[65]:

	YEAR	HAPPY	DATEINTV
4601	1975	2	311
4602	1975	1	318
4603	1975	3	321
4604	1975	2	316
4605	1975	1	308

[67]:
```
#groupy into months.
prev_i = 0
month_list = []
for i in range(199, 1399, 100):
    month_data = dateDf.loc[(dateDf["DATEINTV"] > prev_i) & (dateDf["DATEINTV"]␣
 ↪< i)]
    month_list.append(month_data)
    prev_i = i
```

[176]:
```
months = ["January", "February", "March", "April", "May", "June", "July",␣
 ↪"August", "September", "October", "November", "December"]
monthly_means = []
monthly_num_dps = {}
for i, month in enumerate(month_list):
    monthly_means.append(month.groupby(["YEAR"])["HAPPY"].mean())
    monthly_num_dps[months[i]] = month.shape[0]


print("Number of data points for each month: {}".format(monthly_num_dps))
fig, ax = plt.subplots(figsize=(16,12))
ax.boxplot(monthly_means)
ax.set_xticklabels(months)
plt.show()
```

```
Number of data points for each month: {'January': 27, 'February': 8842, 'March':
14897, 'April': 6802, 'May': 3665, 'June': 2071, 'July': 1076, 'August': 853,
'September': 792, 'October': 514, 'November': 156, 'December': 92}
```

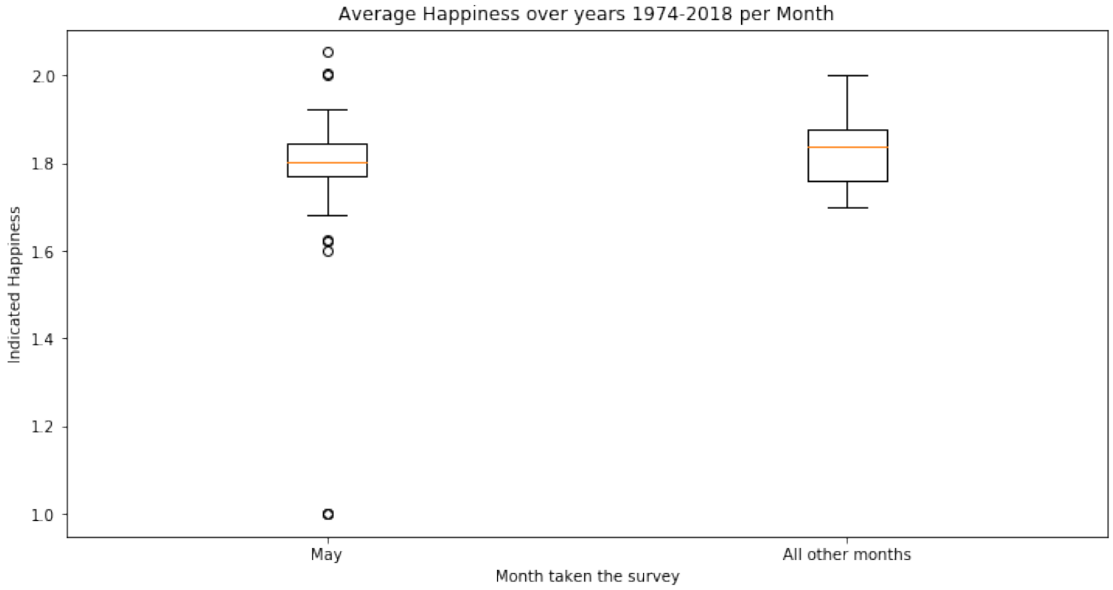


If any stand out, I would say it's the May boxplot. This is because the mean is quite high, and the lowest quartile is stil quit high also. Furthermore, there are not extreme outliers, which makes it seem like there is some good data to be had for that month (and seeing how many data points are available for May, it really seems like good data there). For the other months, especially for December, there isn't so much useful data.

I think it is likely that these deviations are primarily cause by random chance, however when considering students finishing a semester and going into summer break in May, or even graduating, I can definitely see Happiness levels being higher than average there. So let's test that with ANOVA, comparing all of the other months to May.

[81]:
```
notMay = []
for i in range(12):
    if i == 4:
        continue
    notMay = notMay + list(monthly_means[i])

boxes_new = [notMay, monthly_means[4]]

fig, ax = plt.subplots(figsize=(12,6))
ax.boxplot(boxes_new)
ax.set_xticklabels(["May", "All other months"])
ax.set_xlabel("Month taken the survey")
```

```
ax.set_ylabel("Indicated Happiness")
ax.set_title("Average Happiness over years 1974-2018 per Month")
plt.show()


print("Number of data points in first box: {}, num. dps in second box: {}".
 ↪format(len(first11), len(boxes[-1])))
```




```
Number of data points in first box: 198, num. dps in second box: 18
```

This doesn't look very convincing. Let's see what ANOVA says:

[83]:
```
fstat, pval = stats.f_oneway(notMay, monthly_means[4])
print(pval)
```

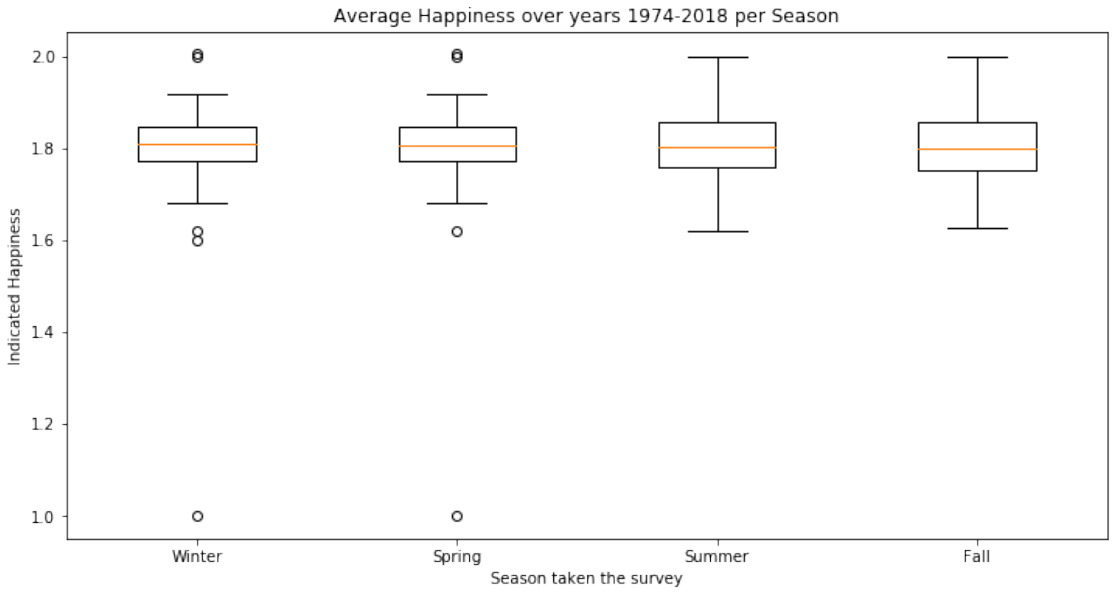
```
0.3075784088146754
```

Yeahhhh, no. Mean happiness in May is not at all significantly different from all of the other months.

What if we compartmentalize into seasons instead?

[84]:
```
seasons = ["Winter", "Spring", "Summer", "Fall"]
seasonal_means = []
for i in range(1,5):
    season = []
    for j in range(4):
        season = season + list(monthly_means[i+j])
    seasonal_means.append(season)
```

```
fig, ax = plt.subplots(figsize=(12,6))
ax.boxplot(seasonal_means)
ax.set_xticklabels(seasons)
ax.set_xlabel("Season taken the survey")
ax.set_ylabel("Indicated Happiness")
ax.set_title("Average Happiness over years 1974-2018 per Season")
plt.show()


# print("Number of data points in first box: {}, num. dps in second box: {}".
 ↪format(len(), len(boxes[-1])))
```



[87]:
```
fstat, pval = stats.f_oneway(*seasonal_means)
print("One-way ANOVA test, f-statistic: {}, p-value: {}".format(fstat, pval))
```

```
One-way ANOVA test, f-statistic: 0.0896749801053436, p-value: 0.9656723408476073
```

And here, too, we see pracitcally no difference besides the number of outliers.

Based on the tests run with the data, it can be concluded that there is not sufficiently statistically significant data to indicate a difference in mean Happiness in different seasons.

1.2.6 ————————- Q8 ————————-

Q8: Using a binary classification model (like logistic regression) combine all these factors to make one synthesized model (including age and health from above as well). Since logistic regression is a binary classification model, youll need

to reduce the HAPPY variable to 2 levels (e.g., HAPPY = 1 or 2 and HAPPY = 3). Explain your decision here and how it may impact the results. What does the model tell you about happiness and what is correlated with it? Make sure to look into the assumptions made by a logistic regression and check these.

First, let's import SKLearn's Logistic regression, train/test split function and confusion matrix function.

[215]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc
```

For the features, we need to include age, health, employment, income, and seasonal data.

For the dependent HAPPY variable, we will reduce the 3 categories into 2. The 3 categories for Happiness are "Not very happy", "Pretty Happy", and "Very Happy". They are all quite different from each other, the first is really a negative indication, the second is neutral, and the third is positive. Ideally, I would like to just drop the neutral category, but this would mean losing a lot of valuable data points. So ultimately, I will combine the first and second categories, because I believe that people that are not happy might be in denial of not being very happy and thus simply choose 'pretty happy', but people that are truly happy and know it will definitely say that they are 'very happy'. For the people that wish to be 'very happy', when they actually are not, I believe 'pretty happy' sounds nice enough that they will be more inclined to choose it than 'very happy'. This choice I'm making may influence the system to have a bias towards not being very happy, since there will be so many more data points in the 'not very happy + pretty happy' group than in the 'very happy' group.

[216]:
```
#before we do anything else, let's drop rows where happiness is undefined
# --- It's hard to predict happiness well if we have some undefined values...

hapDf = fullDf.loc[(fullDf["HAPPY"] != 0) & (fullDf["HAPPY"] < 8)]
```

[217]:
```
#Our dependent variable y, after replacing the variables
happyCats = {1: 0, 2: 0, 3: 1} #represent 'not very happy' and 'pretty happy'␣
 ↪as 0, and 'very happy' as a 1.
y = hapDf["HAPPY"].replace(happyCats).values

# all of the features must have the same number of data points.
# Let's just include all data the way it is, since the number of data points␣
 ↪would decrease significantly
# if we dropped all rows that have at least one of the features with "I don't␣
 ↪know", "Not Applicable", or "No Answer".

#so we need a list of the rows as the X feature and a list of the corresponding␣
 ↪happiness
X = hapDf[["AGE", "HEALTH", "WRKSLF", "RINCOME", "DATEINTV"]].values
```

[218]:
```
#Now that we have our features and our dependent variable, let's divide up our␣
 ↪data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
 ↪random_state=3245)
clf = LogisticRegression(random_state=0).fit(X_train, y_train)
```

```
y_pred = clf.predict(X_test)
print("The logistic regression's score is: {}".format(clf.score(X_test,␣
 ↪y_test)))
```

```
The logistic regression's score is: 0.8684855082324126
```

Looks like we got a logistic regression score of 0.868! Not too bad, although I am worried about the bias towards less happiness concerns me. Let's look at the predicted values.

[219]:
```
print(y_pred[:200])
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Out of 200 people, the regression predicted 1 person to be 'very happy'. That is not very useful. Let's look at predicted and true value pairs to get a better feel for it.

[220]:
```
print([(y_pred[i], y_test[i]) for i in range(100)])
```

```
[(0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0),
(0, 0), (0, 1), (0, 1), (0, 0), (0, 1), (0, 0), (0, 0), (0, 0), (0, 1), (0, 0),
(0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 1), (0, 0), (0, 0), (0, 1),
(0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0),
(0, 0), (0, 0), (0, 0), (0, 0), (0, 1), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0),
(0, 0), (0, 0), (0, 1), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0),
(0, 0), (0, 0), (0, 0), (0, 1), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0),
(0, 1), (0, 0), (0, 0), (0, 0), (0, 0), (0, 1), (0, 0), (0, 0), (0, 1), (0, 0),
(0, 0), (0, 0), (0, 0), (0, 1), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0),
(0, 0), (0, 0), (0, 0), (0, 1), (0, 0), (0, 0), (0, 1), (0, 0), (0, 1), (0, 0)]
```

As we can see, there are significantly many true values that are ones, but are predicted to be zeros.

Another way for us to check is using the Area Under the Curve (AUC).

[221]:
```
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(false_positive_rate, true_positive_rate)
roc_auc
```

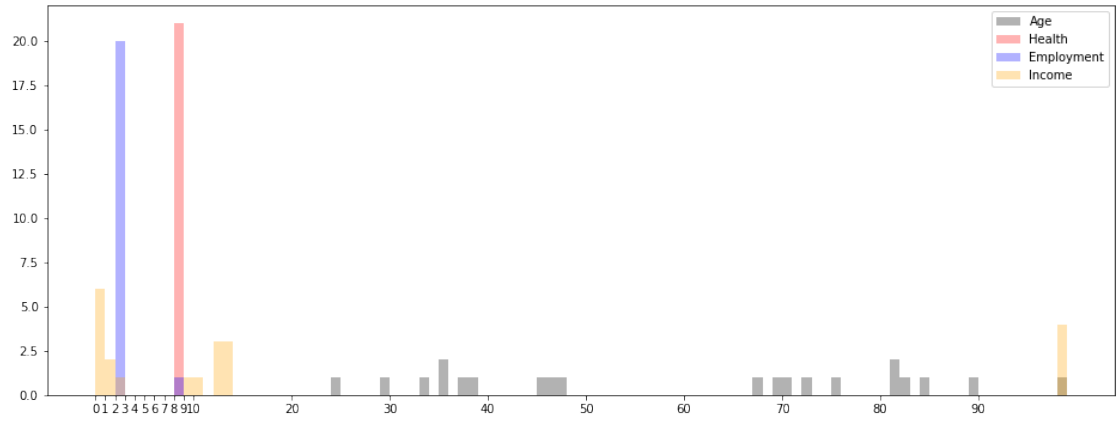
[221]:
```
0.501565683315083
```

An area under the curve of 50% is as good as guessing…

Despite this problem, we can still look at what the features are set when the model does predict a 1.

[222]:
```
colors = ['black', 'red', 'blue', 'orange', 'purple']
labels = ["Age", "Health", "Employment", "Income", "Date survey taken"]
```

```
feature_vals = []
for i,predicted_val in enumerate(y_pred):
    if predicted_val == 1:
        feature_vals.append(X_test[i])


fig, ax = plt.subplots(figsize=(16, 6))
median_feat_vals = []
for i in range(5):
    feat = [feats[i] for feats in feature_vals]
    median_feat_vals.append(feat)

    #DONT PLOT DATE TAKEN, IT MESSES WITH THE SCALE
    if i == 4:
        continue

    ax.hist(feat, color = colors[i], label = labels[i], alpha = 0.3, bins =␣
 ↪range(min(feat), max(feat)+1, 1))
#     print(featur)
median_feat_vals = [np.median(i) for i in median_feat_vals]

ax.set_xticks(list(range(0, 10, 1)) + list(range(10, 100, 10)))
plt.legend()
plt.show()
```



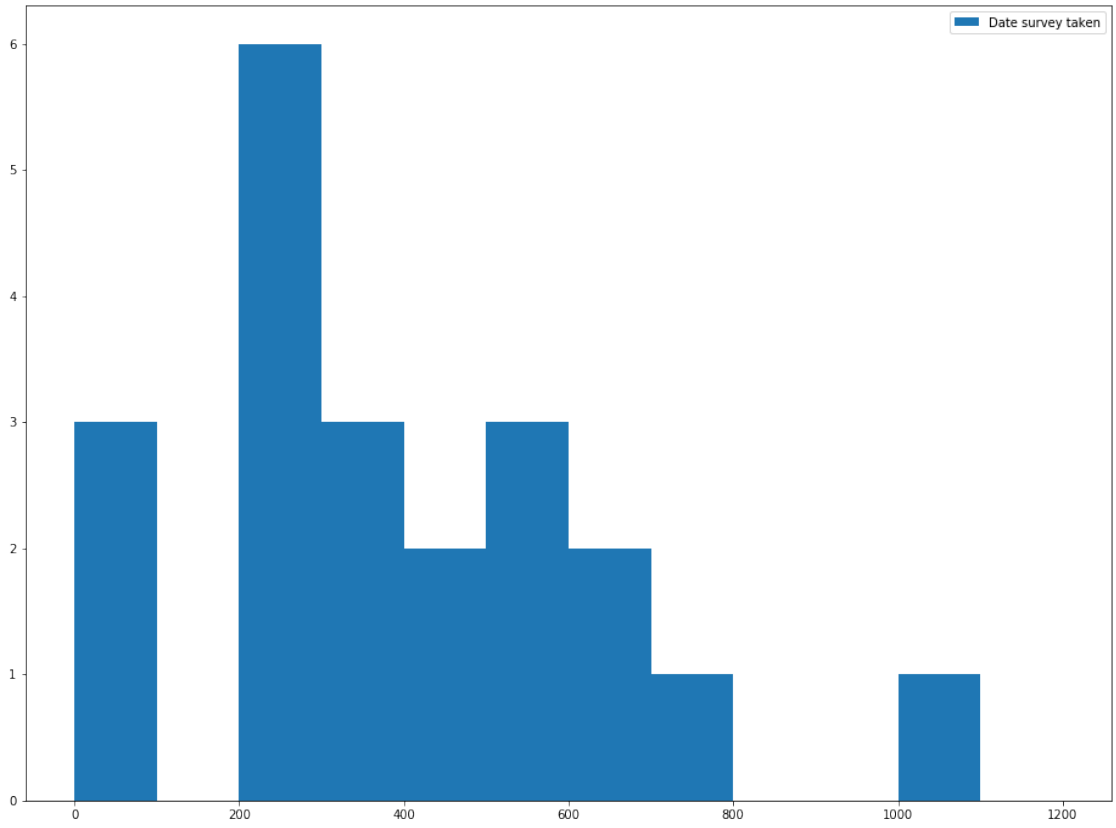
Although I'll admit it isn't the prettiest plot, it does show a few things. The linear regression seems to look at the data in this way:

- Age mainly doesn't matter, since it is so widely distributed.
- Health doesn't matter much, since practically all of them have a value of 8 (meaning "Don't know').
- Employment matters a lot, since most of the features for which the regression predicted yes indicate the person works for someone else

- Income matters a little bit, since there is a visible increase of counts as income increases, but the peaks are at the unknown data points

Lastly, let's look at the date the survey was taken:

[223]:
```
fig, ax = plt.subplots(figsize=(16,12))
ax.hist([feats[4] for feats in feature_vals], label = labels[4], bins =␣
 ↪range(0, 1300, 100))
plt.legend()
plt.show()
```



[224]:
```
print("Number of data points for each month: {}".format(monthly_num_dps))
```

```
Number of data points for each month: {'January': 27, 'February': 8842, 'March':
14897, 'April': 6802, 'May': 3665, 'June': 2071, 'July': 1076, 'August': 853,
'September': 792, 'October': 514, 'November': 156, 'December': 92}
```

Looks like there is an influx in counts during February (in between 200 and 300). The model seems to choose mostly those, however this may also be due to random chance since as we have found out while testing hypothesis 3, there is a higher number of tests taken over the years in February, March and April than in all other months. March and April are interesting, it looks as if the regression had a bias against them, since even though they had a lot of data points, February

has a far higher count. But really there aren't enough predicted values equal to 1 to make much of a statement on how the regression makes a choice.

Overall, it doesn't look like we can trust the model much. It has a very strong bias against labeling people 'very happy', and if we look at the inputs for which it did return 'very happy' and the individual features they are comprised of, there isn't much correlation to the data exploration and statistical tests we have done on them throughout the notebook and the regressions choices.

Let's go through the assumption a linear regression makes to (hopefully) gain a better understanding of the choices it made. According to this website, the main assumptions are:

– NOT MY OWN WORDS –

1) Logistic regression does not require a linear relationship between the dependent and independent variables.

2) The error terms (residuals) do not need to be normally distributed.

3) Homoscedasticity is not required.

4) The dependent variable in logistic regression is not measured on an interval or ratio scale.

Other assumptions:

1) Binary logistic regression requires the dependent variable to be binary and ordinal logistic regression requires the dependent variable to be ordinal.

2) Logistic regression requires the observations to be independent of each other. In other words, the observations should not come from repeated measurements or matched data.

3) Logistic regression requires there to be little or no multicollinearity among the independent variables. This means that the independent variables should not be too highly correlated with each other.

4) Logistic regression assumes linearity of independent variables and log odds. Although this analysis does not require the dependent and independent variables to be related linearly, it requires that the independent variables are linearly related to the log odds.

5) Logistic regression typically requires a large sample size. A general guideline is that you need at minimum of 10 cases with the least frequent outcome for each independent variable in your model. For example, if you have 5 independent variables and the expected probability of your least frequent outcome is .10, then you would need a minimum sample size of 500 (10*5 / .10).

– MY OWN WORDS AGAIN –

Following these notes, we can see that although the data does not have a linear relationship between the dependent and independent variables, and is not homoscedastic (meaning the sequence of random variables, in our case the features, do not all have the same finite variance). Furthermore, the data correctly has a binary dependent variable for binary logistic regression, the data points are independent from each other, the features are not highly correlated with each other. However we do not have enough data points according to the calculation shown in assumption 5 above, and I also do not think that the features are lineraly related to the log odds.

Overall, it does seem like linear regression could work here, sure there isn't enough data available to really predict happiness well, and the features likely are not lineraly related to the log odds, but overal, the other assumptions are met perfectly. I believe we could expect the linear regression to do an okay job, however looking at the predicted y labels, it becomes clear that it really cannot correctly predict a person to be 'very happy' based on the features given.

This leads me to believe that the fault of the prediction lies mainly in the features chosen.

1.3 Extra Credit

Finally, try a multi-class classification model like a random forest, neural network, or support vector machine. How well or poorly does this work? Which variables contribute most to your model (i.e., can you explain what contributes most to happiness using this model?).

Finally, let's see what a Random Forest Classifier can do with the features! For this classifier, I would really prefer it be multiclass, however it will be difficult to find good parameters, so let's actually stick to the binary task. This also means we can use the same variables as before.

[225]:
```
from sklearn.ensemble import RandomForestClassifier
```

Let's use the default random forest classifier from SKLearn and see how it performs.

[227]:
```
clf = RandomForestClassifier()
clf.fit(X_train, y_train)
print("Feature importances: {}".format(clf.feature_importances_))
y_pred = clf.predict(X_test)

false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(false_positive_rate, true_positive_rate)

print("The AUC for the random forest is: {}".format(roc_auc))
```

```
Feature importances: [0.35770147 0.06288027 0.02618056 0.08920384 0.46403385]
The AUC for the random forest is: 0.5148347966090129
```

So we get an Area Under the Curve of 0.515, which is still fairly bad, BUT it is at least better than the linear regression, and we are only using the default parameters! Let's see if we can make it better, though:

[262]:
```
n_estimators =list(np.arange(1, 10, 1)) + list(np.arange(10, 30, 2))
train_AUCs = []
test_AUCs = []

for est in n_estimators:
    clf = RandomForestClassifier(n_estimators=est)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,␣
 ↪y_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    test_AUCs.append((roc_auc, est))

best_estimator = -1
best_auc = 0.5
for curAUC, estimator in test_AUCs:
    if curAUC > best_auc:
        best_auc = curAUC
        best_estimator = estimator
```

```
print("Best estimator is {}, and corresponding best AUC is {}".
 ↪format(best_estimator, best_auc))
```

```
Best estimator is 5, and corresponding best AUC is 0.5265268945954114
```

[263]:
```
test_AUCs = []
for max_depth in range(1, 50):
    clf = RandomForestClassifier(n_estimators = best_estimator,␣
 ↪max_depth=max_depth)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,␣
 ↪y_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    test_AUCs.append((roc_auc, max_depth))

best_depth = -1
best_auc = 0.5
for curAUC, depth in test_AUCs:
    if curAUC > best_auc:
        best_auc = curAUC
        best_depth = depth

print("Using the best estimator, the best max-depth is {}, and corresponding␣
 ↪best AUC is {}".format(best_depth, best_auc))
```

```
Using the best estimator, the best max-depth is 25, and corresponding best AUC
is 0.5274233404926614
```

Based on these calculations, it doesn't seem like the Random Forest Classifier can do much better...

Regardless, let's look at the feature importances:

[268]:
```
clf = RandomForestClassifier(n_estimators=best_estimator, max_depth=best_auc)
clf.fit(X_train, y_train)
print("Feature importances: {}".format(clf.feature_importances_))
y_pred = clf.predict(X_test)
print(pd.Series(y_pred).unique())
```

```
Feature importances: [0. 0. 0. 0. 0.]
[0]
```

So the random forest classifier actually learns to just choose 0 for every single data point...

Based on this, unfortunately you really cannot learn which is the best feature to use for predicting happiness.

Let's try again, but this time using only rows that have all of the data available.

[269]:
```
hapDf = hapDf.loc[(hapDf["HEALTH"] != 0) & (hapDf["HEALTH"] < 8)]
hapDf = hapDf.loc[(hapDf["WRKSLF"] != 0) & (hapDf["WRKSLF"] < 8)]
hapDf = hapDf.loc[(hapDf["RINCOME"] != 0) & (hapDf["RINCOME"] < 12)]
hapDf = hapDf.loc[(hapDf["DATEINTV"] != 0) & (hapDf["DATEINTV"] < 8)]
```

[270]:
```
print(hapDf.shape)
```

```
(0, 6108)
```

Unfortunately, we can't do that, as there are no rows for which all of the data are available.

I suppose this is where it all ends. There isn't one specific, clear answer to the question that is if we can figure out which feature the classifier takes into account the most.

[ ]:
```

```