World Happiness

An empirical study with machine learning applications

Giovanni Ravelli, HarvardX - Data Science

6/10/2020

Executive Summary

The World Happiness Report is a "landmark survey of the state of global happiness", as stated by Gallup Research who every year performs this global poll involving more than 150 countries representing approximately 98% of the world's population. The first report was published in 2012 and was assembled in Thimphu in July 2011, pursuant to a Bhutanese resolution that invited national governments to "give more importance to happiness and well-being in determining how to achieve and measure social and economic development." The report continues to gain global recognition as governments, organizations and civil society increasingly use happiness indicators to inform their policy-making decisions.

In the World Happiness Report, the happiness ranking is based on answers to the evaluation question asked in the poll. This question, known as the **Cantril ladder**, asks respondents to think of a ladder with the best possible life being a 10 and the worst possible life being a 0, and to rate their own current lives on that scale.

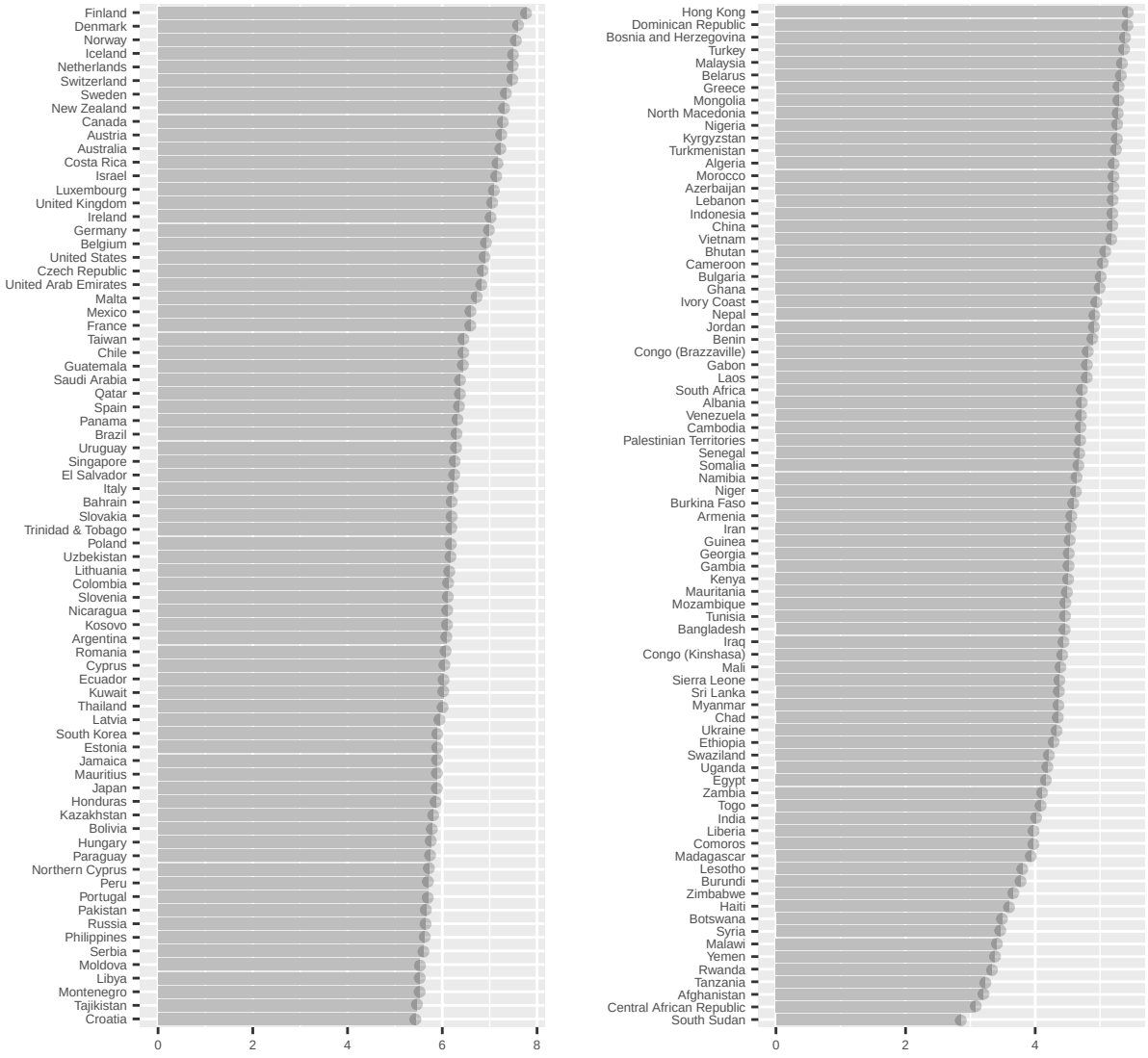
The authors, subsequently, use six variables to explain happiness variability across countries in the form of a **Happiness Score**. The **six variables** used to predict happiness are: economic production, social support, life expectancy, freedom, absence of corruption, and generosity.

They benchmark this evaluation against Dystopia (in contrast to Utopia), an imaginary country that has the world's least-happy people. The purpose in establishing Dystopia is to have a benchmark against which all countries can be favorably compared (no country performs more poorly than Dystopia) in terms of each of the six key variables.

As stated, the happiness ranking is not based on any index of these six factors, but depends only on the average Cantril ladder scores reported by the respondents. The six variables are used to explain the variation of happiness across countries. According to the authors, taken together, these six variables explain three-quarters of the cross-country variation.

In this current work we analyze the available data and use different machines learning algorithms to predict the happiness score and understand which factors are most important to pursue happiness in a society.

Contents



Database overview

The World Happiness Report use publicly available data from the **Gallup World Poll**.

We are using the 2019 dataset, a "class data-frame" dataset which is comprised of 156 observations (countries) and 9 variables: the 6 predictors described previously, in addition to Rank, Country and Score. The latter is what we will try to predict using different prediction models. The happiness score and the six variables (*predictors*) are all class "numeric".

```
## [1] "data.frame"
```

```
## 'data.frame':    156 obs. of  9 variables:
##  $ Overall.rank                : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Country                     : Factor w/ 156 levels "Afghanistan",..: 44 37 106 58 99 134 133 100 1
##  $ Score                       : num  7.77 7.6 7.55 7.49 7.49 ...
##  $ GDP.per.capita              : num  1.34 1.38 1.49 1.38 1.4 ...
##  $ Social.support              : num  1.59 1.57 1.58 1.62 1.52 ...
##  $ Healthy.life.expectancy     : num  0.986 0.996 1.028 1.026 0.999 ...
##  $ Freedom.to.make.life.choices: num  0.596 0.592 0.603 0.591 0.557 0.572 0.574 0.585 0.584 0.532 ..
##  $ Generosity                  : num  0.153 0.252 0.271 0.354 0.322 0.263 0.267 0.33 0.285 0.244 ...
##  $ Perceptions.of.corruption   : num  0.393 0.41 0.341 0.118 0.298 0.343 0.373 0.38 0.308 0.226 ...
```

Some elementary statistics are included below: for Score and each variable, min/max values, mean/median and 1st/3rd quantile values. (To note that Somalia's GDP per capita is zero, presumably because not provided).
The gap between median and mean GDP per capita is likely an indication of wealth inequality across countries. Similar spread between mean/median is found in other dimensions such as Corruption, Life Expectancy, Freedom. We will look into this with some more analytics soon.

```
##   Overall.rank           Country        Score       GDP.per.capita
##  Min.   :  1.00   Afghanistan:  1   Min.   :2.853   Min.   :0.0000
##  1st Qu.: 39.75   Albania    :  1   1st Qu.:4.545   1st Qu.:0.6028
##  Median : 78.50   Algeria    :  1   Median :5.380   Median :0.9600
##  Mean   : 78.50   Argentina  :  1   Mean   :5.407   Mean   :0.9051
##  3rd Qu.:117.25   Armenia    :  1   3rd Qu.:6.184   3rd Qu.:1.2325
##  Max.   :156.00   Australia  :  1   Max.   :7.769   Max.   :1.6840
##                   (Other)    :150
##  Social.support  Healthy.life.expectancy Freedom.to.make.life.choices
##  Min.   :0.000   Min.   :0.0000          Min.   :0.0000
##  1st Qu.:1.056   1st Qu.:0.5477          1st Qu.:0.3080
##  Median :1.272   Median :0.7890          Median :0.4170
##  Mean   :1.209   Mean   :0.7252          Mean   :0.3926
##  3rd Qu.:1.452   3rd Qu.:0.8818          3rd Qu.:0.5072
##  Max.   :1.624   Max.   :1.1410          Max.   :0.6310
##
##    Generosity     Perceptions.of.corruption
##  Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.1087   1st Qu.:0.0470
##  Median :0.1775   Median :0.0855
##  Mean   :0.1848   Mean   :0.1106
##  3rd Qu.:0.2482   3rd Qu.:0.1412
##  Max.   :0.5660   Max.   :0.4530
##
```

Data Wrangling

The dataset is relatively simple and largely clean.

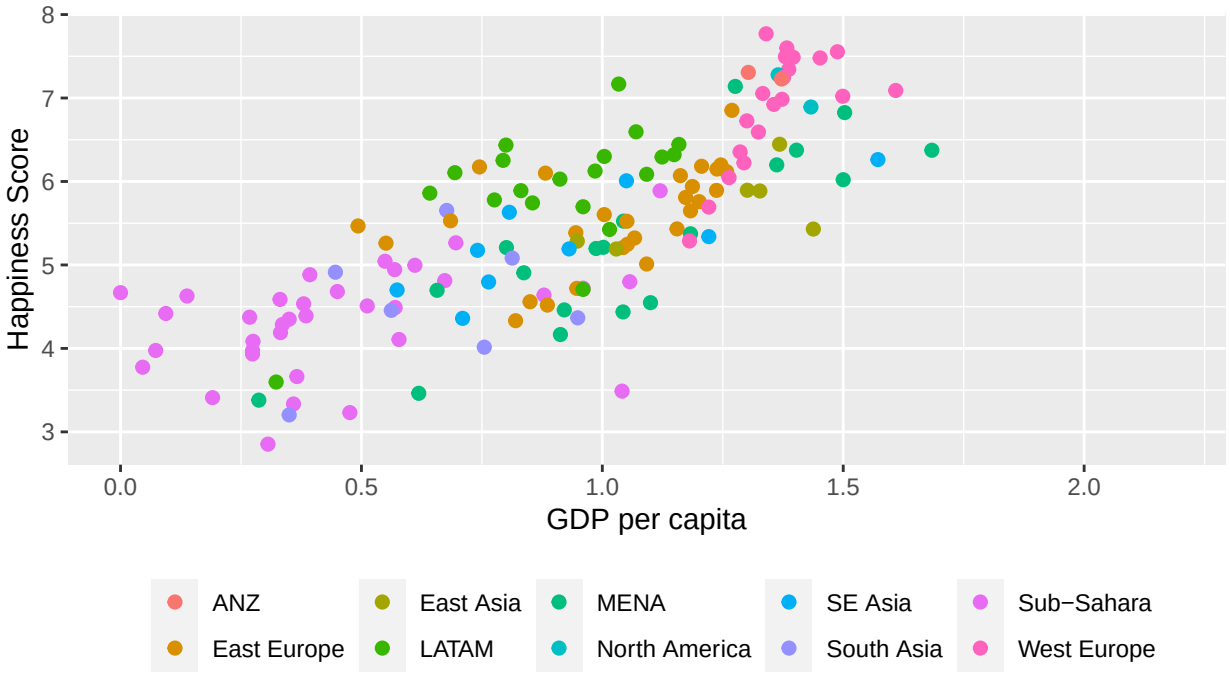
We run some data wrangling in order to:
1. Rename columns so are more readable in charts;
2. Add a "Region" column extracted from the Happiness Ranking 2016;
3. Shorten the variables' and regions' names to improve data visualization in charts and tables;
4. Import the 2016 happiness score for time comparability;
5. Check, clean and fill-in any residual missing value or "NA" and;
6. Re-order columns in the data frame.

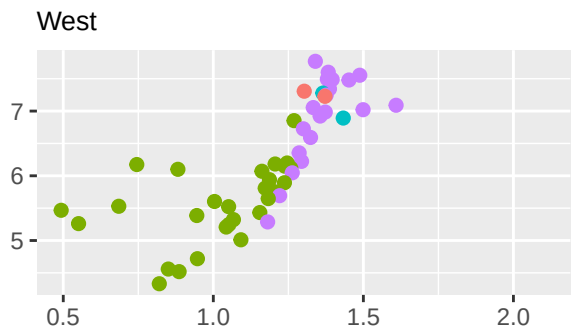
After the above procedure is performed, we obtain the following outcome (showing only top 30 countries):

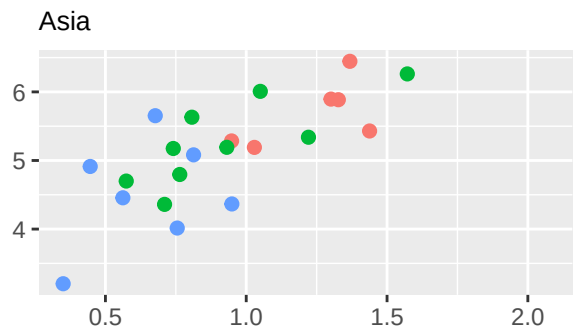
Rank	Country	Region	Score_2016	Score	Economy	Family	Health	Freedom	Generosity	Corruption
1	Finland	West Europe	7.413	7.769	1.340	1.587	0.986	0.596	0.153	0.393
2	Denmark	West Europe	7.526	7.600	1.383	1.573	0.996	0.592	0.252	0.410
3	Norway	West Europe	7.498	7.554	1.488	1.582	1.028	0.603	0.271	0.341
4	Iceland	West Europe	7.501	7.494	1.380	1.624	1.026	0.591	0.354	0.118
5	Netherlands	West Europe	7.339	7.488	1.396	1.522	0.999	0.557	0.322	0.298
6	Switzerland	West Europe	7.509	7.480	1.452	1.526	1.052	0.572	0.263	0.343
7	Sweden	West Europe	7.291	7.343	1.387	1.487	1.009	0.574	0.267	0.373
8	New Zealand	ANZ	7.334	7.307	1.303	1.557	1.026	0.585	0.330	0.380
9	Canada	North America	7.404	7.278	1.365	1.505	1.039	0.584	0.285	0.308
10	Austria	West Europe	7.119	7.246	1.376	1.475	1.016	0.532	0.244	0.226
11	Australia	ANZ	7.313	7.228	1.372	1.548	1.036	0.557	0.332	0.290
12	Costa Rica	LATAM	7.087	7.167	1.034	1.441	0.963	0.558	0.144	0.093
13	Israel	MENA	7.267	7.139	1.276	1.455	1.029	0.371	0.261	0.082
14	Luxembourg	West Europe	6.871	7.090	1.609	1.479	1.012	0.526	0.194	0.316
15	United Kingdom	West Europe	6.725	7.054	1.333	1.538	0.996	0.450	0.348	0.278
16	Ireland	West Europe	6.907	7.021	1.499	1.553	0.999	0.516	0.298	0.310
17	Germany	West Europe	6.994	6.985	1.373	1.454	0.987	0.495	0.261	0.265
18	Belgium	West Europe	6.929	6.923	1.356	1.504	0.986	0.473	0.160	0.210
19	United States	North America	7.104	6.892	1.433	1.457	0.874	0.454	0.280	0.128
20	Czech Republic	East Europe	6.596	6.852	1.269	1.487	0.920	0.457	0.046	0.036
21	United Arab Emirates	MENA	6.573	6.825	1.503	1.310	0.825	0.598	0.262	0.182
22	Malta	West Europe	6.488	6.726	1.300	1.520	0.999	0.564	0.375	0.151
23	Mexico	LATAM	6.778	6.595	1.070	1.323	0.861	0.433	0.074	0.073
24	France	West Europe	6.478	6.592	1.324	1.472	1.045	0.436	0.111	0.183
25	Taiwan	East Asia	6.379	6.446	1.368	1.430	0.914	0.351	0.242	0.097
26	Chile	LATAM	6.705	6.444	1.159	1.369	0.920	0.357	0.187	0.056
27	Guatemala	LATAM	6.324	6.436	0.800	1.269	0.746	0.535	0.175	0.078
28	Saudi Arabia	MENA	6.379	6.375	1.403	1.357	0.795	0.439	0.080	0.132
29	Qatar	MENA	6.375	6.374	1.684	1.313	0.871	0.555	0.220	0.167
30	Spain	West Europe	6.361	6.354	1.286	1.484	1.062	0.362	0.153	0.079

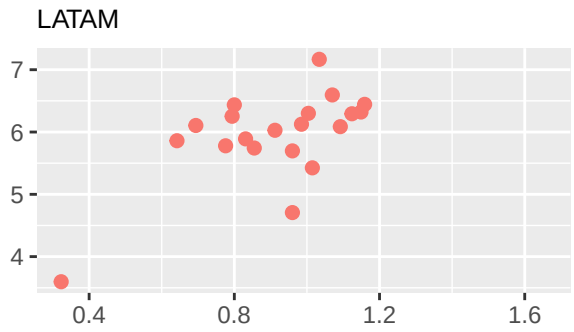
Data Visualization

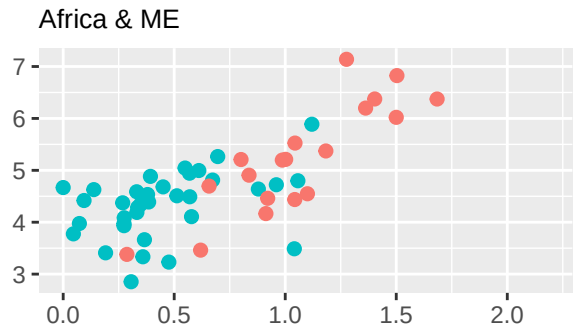
These scatter plots allow to visualize two main results: 1) a positive relationship (correlation) between happiness and GDP per capita, such correlation being strongest in the Western world and weakest in South America, with Asia somewhere in between; 2) happiness is highest in Western Europe and lowest in Africa.



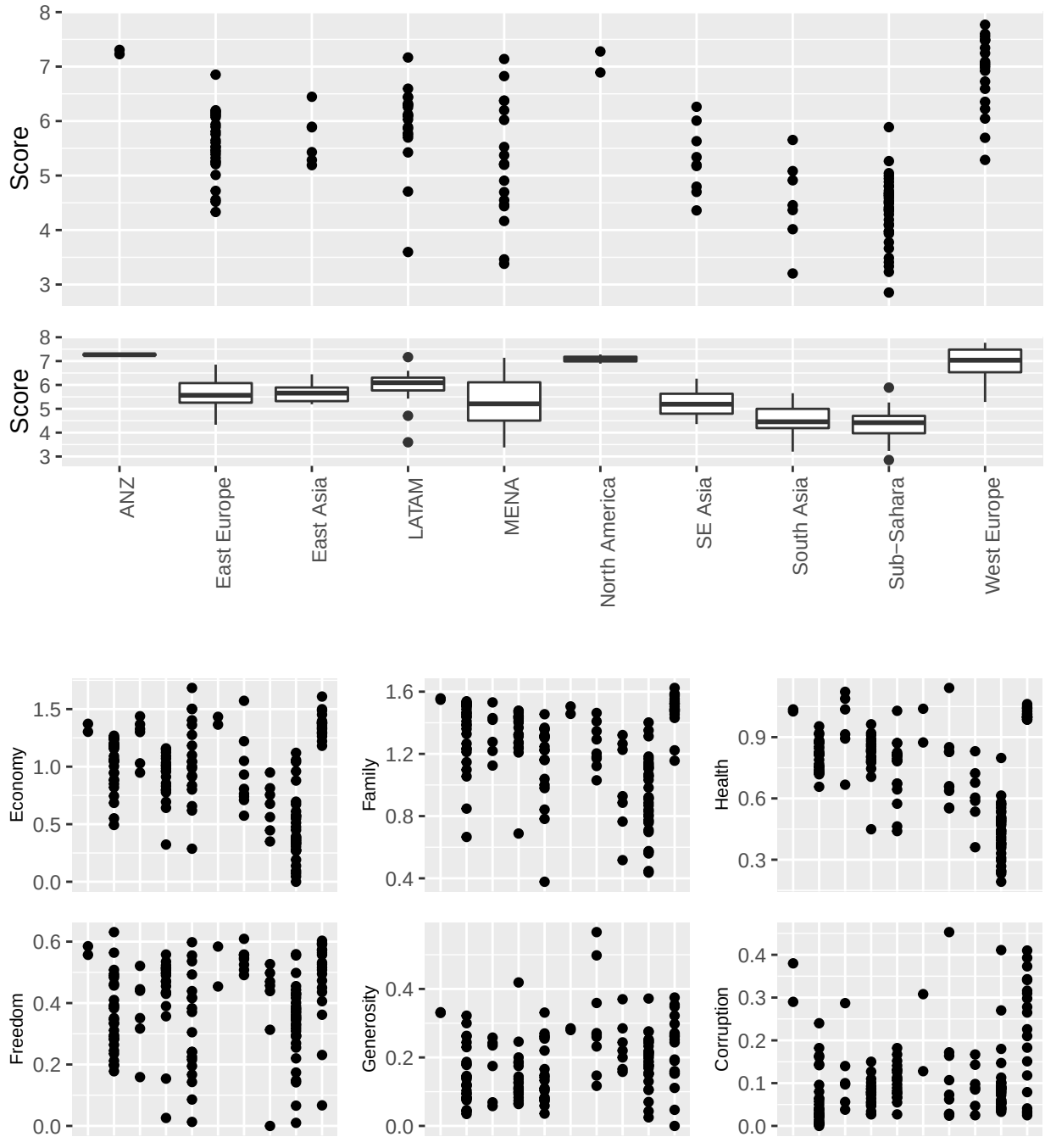




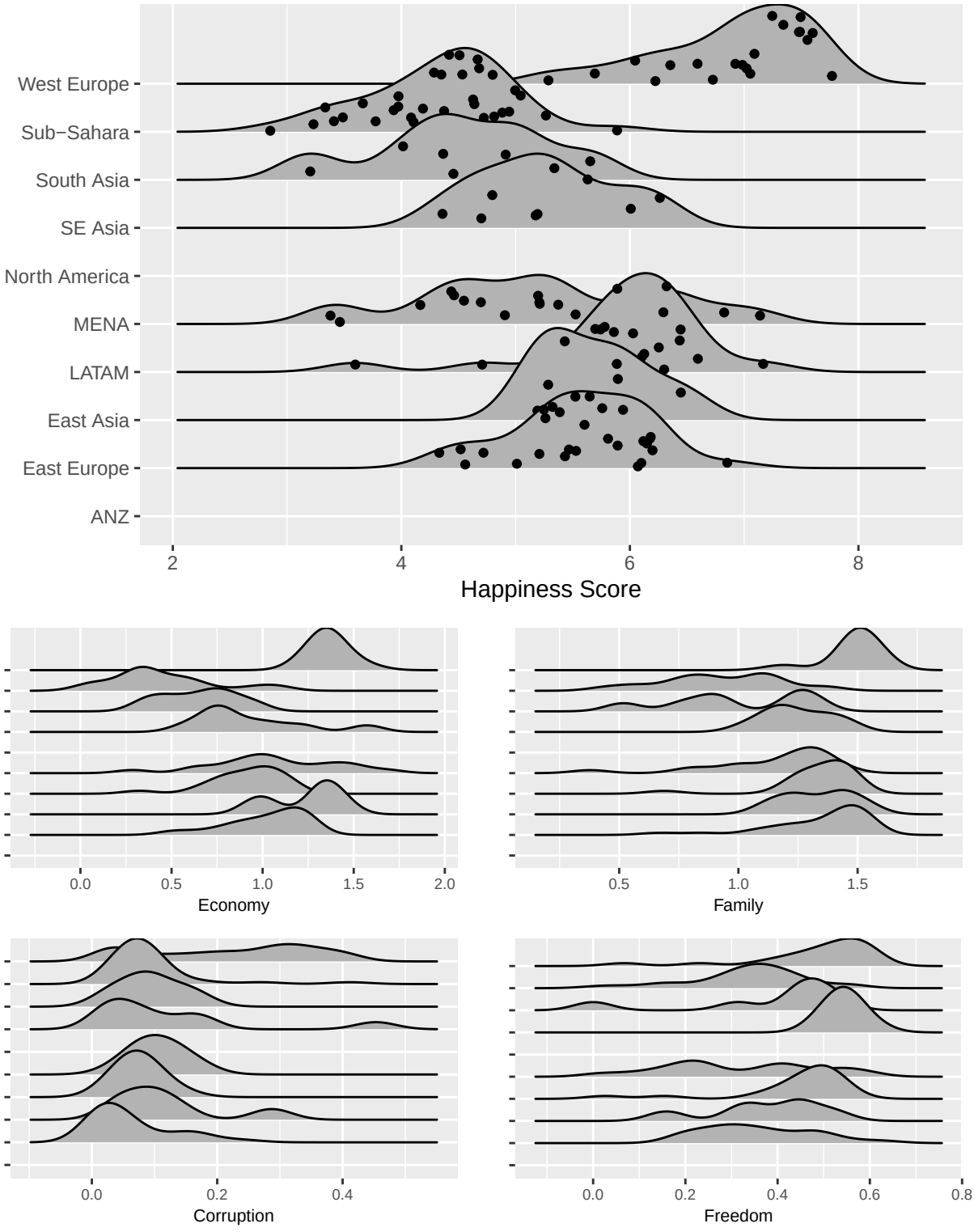




Furthermore below, the q-plot and box-plot show the distribution of the happiness score within each region. While as expected Western Europe, North America and Australia/New Zealand show distributions that are relatively compact in the upper range of the score, we observe higher dispersion across Middle East and North Africa (MENA), while Sub-Saharan countries are fairly concentrated at the bottom end of the range, save for a couple of outliers. Latin America also features a few outliers in what is otherwise a rather concentrated distribution. A q-plot of the six predictors provide for similar results.

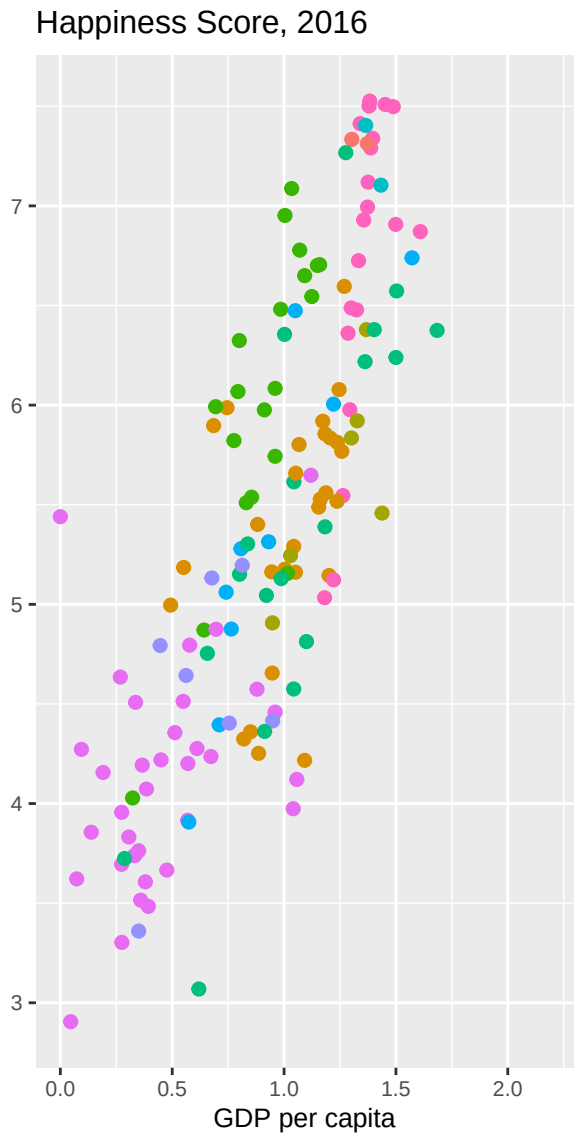


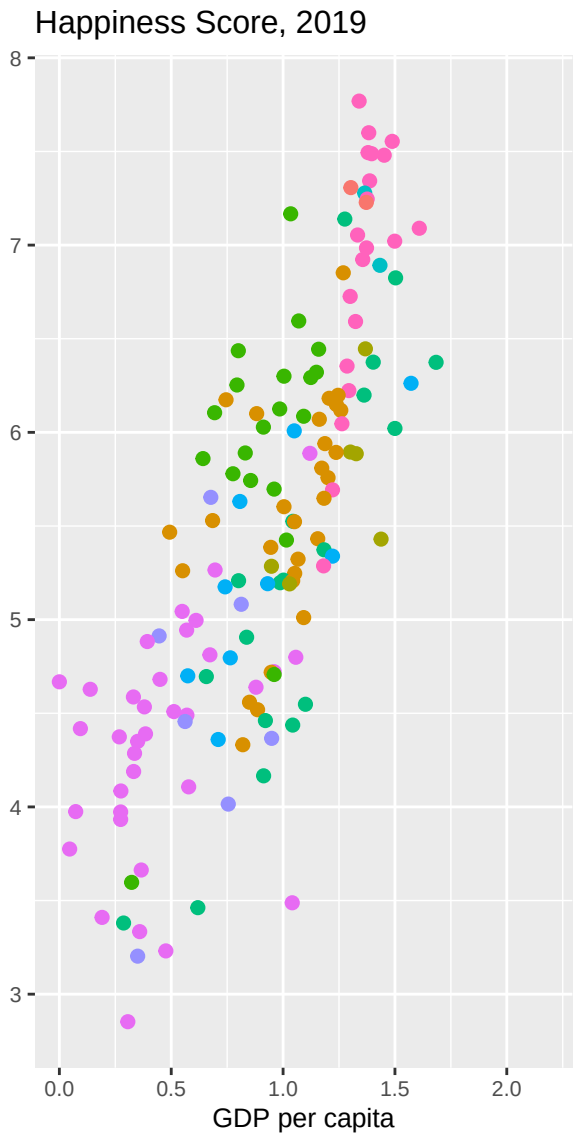
Similarly below, we visualize some density plots picturing the distribution of the happiness score and of some predictors within each Region. To note that for ANZ and North America we do not have enough observations to form a density. One thing to notice: trust in governments ("Corruption") is low in most regions around the world.

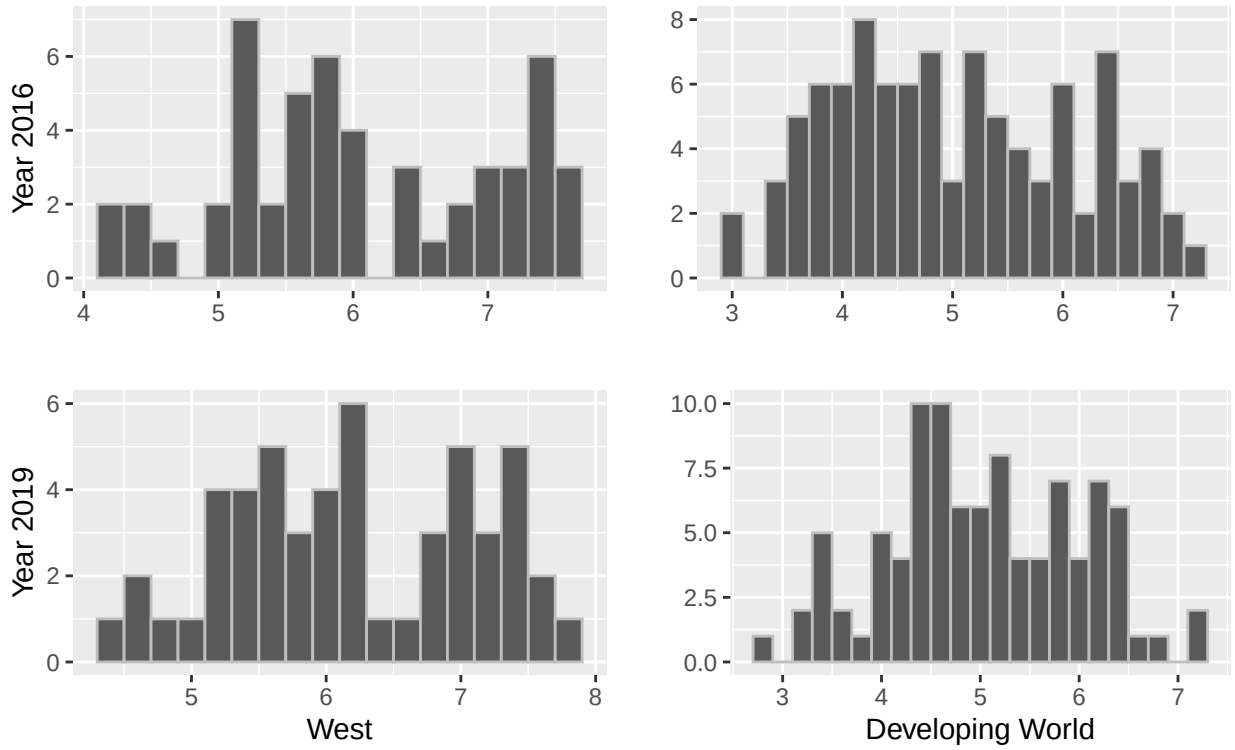


Time comparison of world happiness

Looking at the data across time, we conclude rapidly that happiness has not changed much in the past 3 years. This is no surprise, given that the key determinants of happiness in a society seem intimately related with structural factors such as the economy, or the inclusiveness, quality and strength of institutions, measured among other things by freedom, trust in government, as well as the fabric of a society such as family and education.





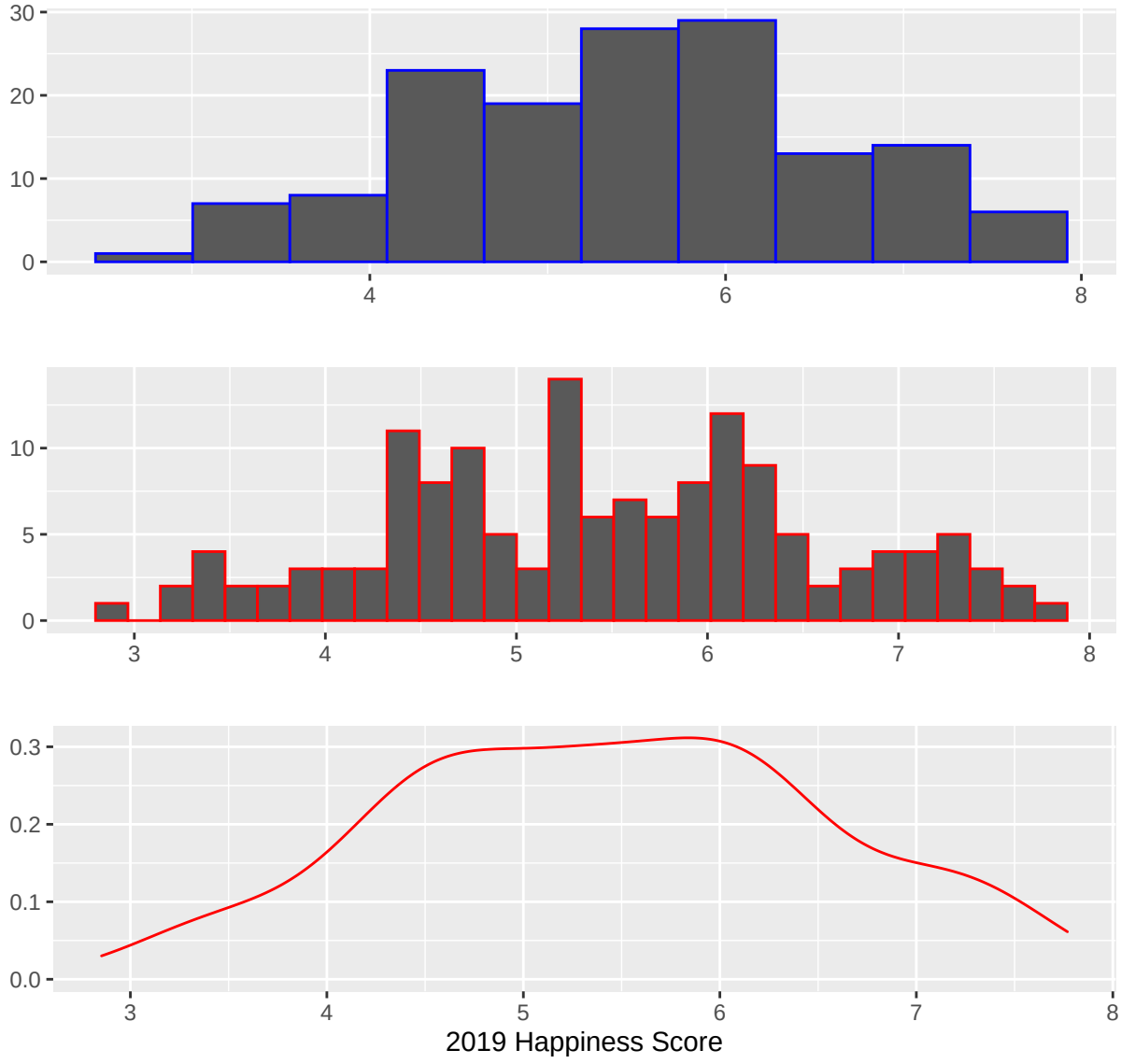

Year 2016
Year 2019
West
Developing World

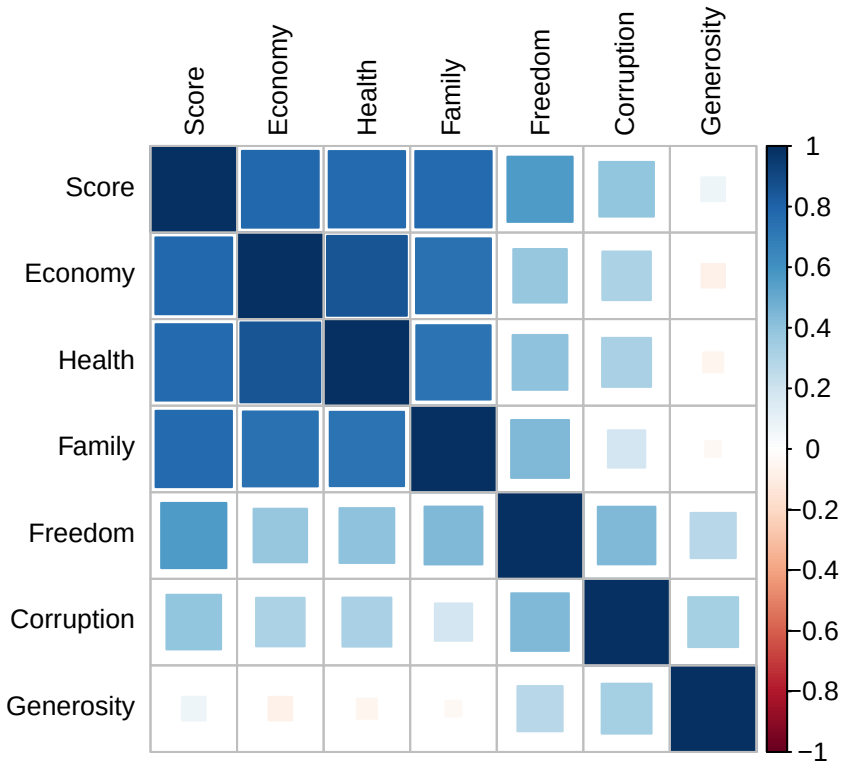

Happiness Score, 2016
Happiness Score, 2019
ANZ
East Europe
East Asia
LATAM
MENA
North America
SE Asia
South Asia
Sub−Sahara
West Europe

Exploratory Data Analysis

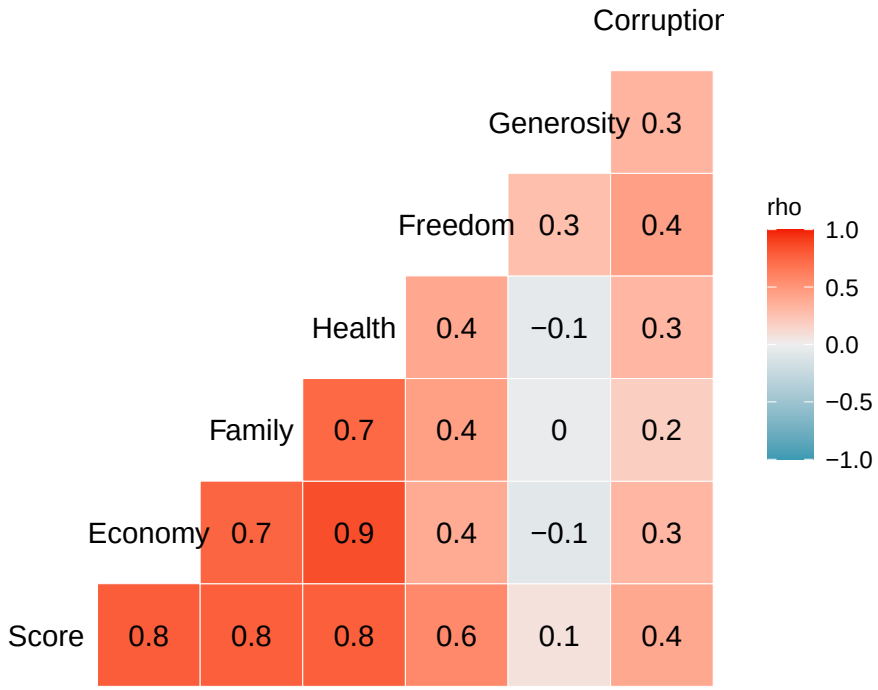
So far we have focused on **univariate discrete analysis** to gain more insight from the data, and looked at some of their properties such as frequency/distribution (in the form of histograms and density plots), or simple measures of central tendency (mean and median) and variability/dispersion (outliers, standard deviation). The dataset is made of a finite number of observations, therefore we call this a discrete analysis. Furthermore, we have considered one variable at a time, we call this univariate analysis.

We are now adding a bit more sophistication along two directions: 1) we use **bivariate/multivariate analysis** to understand the relationship between more variables at the same time ; 2) we move from discrete to continuous, to relax some assumptions on the distribution of the data and, eventually, set the framework for our machine learning model, and particularly for our regression analysis. The charts below provide a sense of what we try to do.





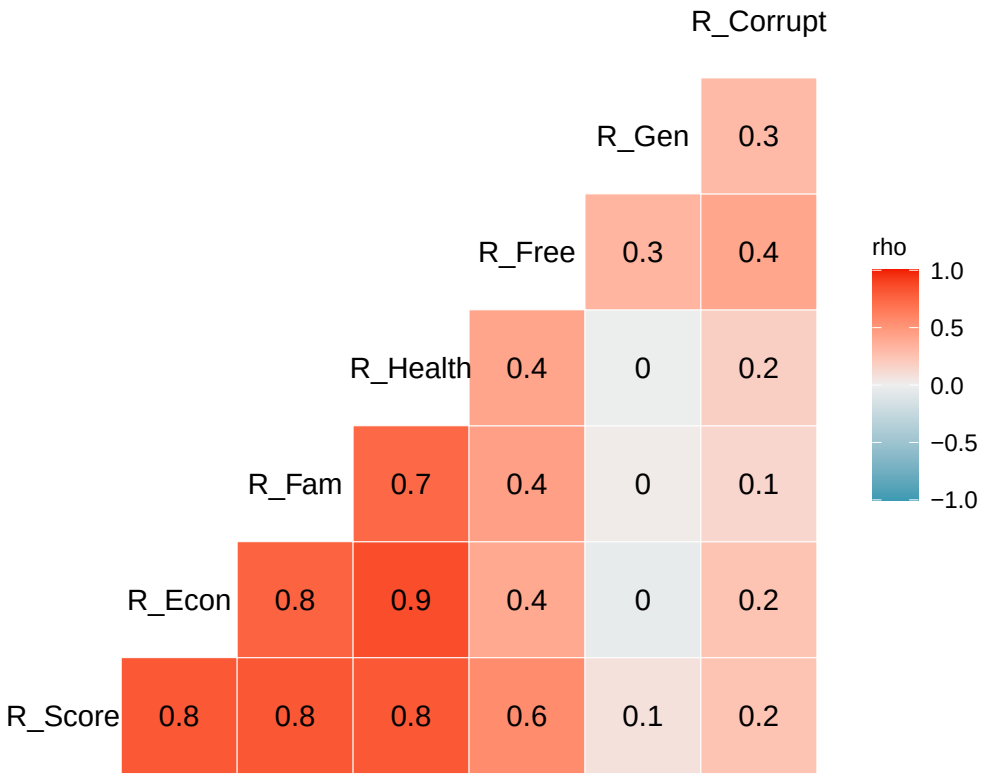
The correlation matrix pictured above shows two important results, among others: 1) the happiness score is positively correlated with all predictors and, all predictors are correlated with each other, with the exception perhaps of "Generosity" where correlation is more nuanced; 2) Economy, Health and Family are stronger determinants of happiness. Here is another way to visualize this:



Association is not causation: a short digression on spurious correlation and confounding

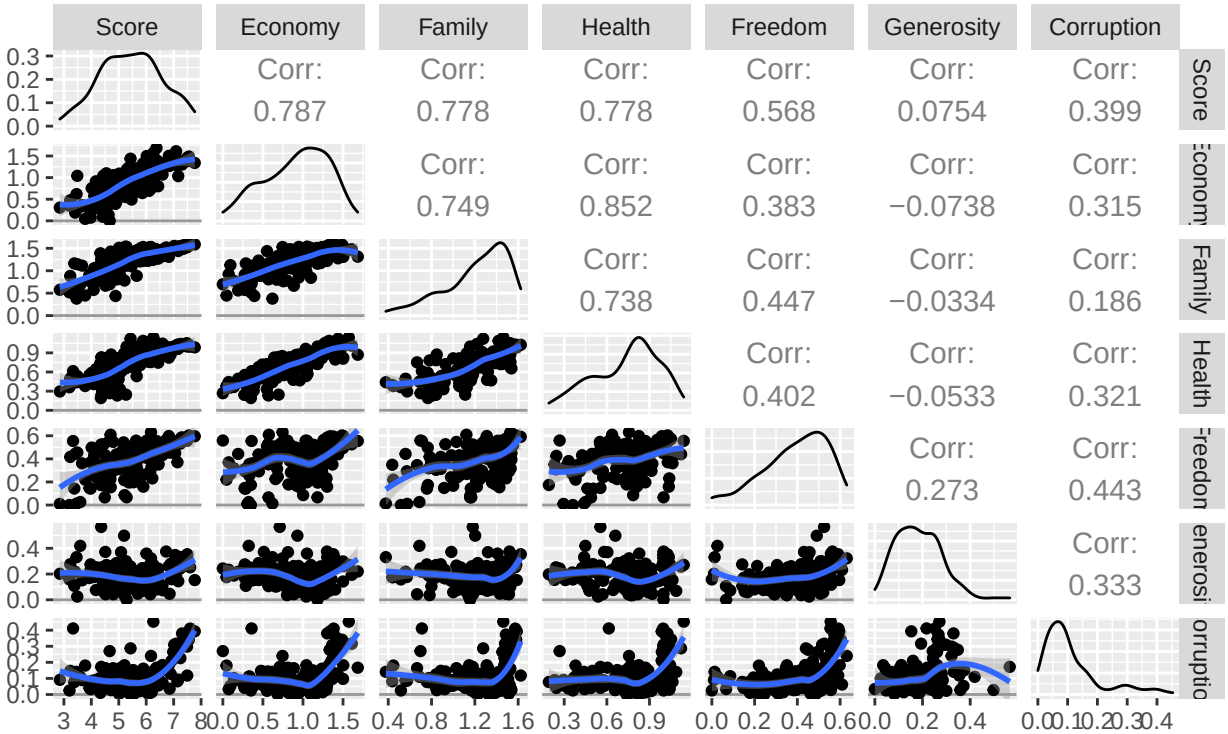
We noted in the preceding section a strong correlation between the Happiness Score and most variables used as predictors. A critical way to challenge these results is to argue that, "correlation is not causation", i.e. the state of the economy, for example, albeit strongly correlated with happiness, does not necessarily create happiness. These two distributions may simply move together. While this observation defies logic and, most importantly, a wealth of empirical studies, we will use this example to test whether this could be the case. Note that, since we are not sampling (we are using the entire population), we can exclude in the first instance any attempt of data dredging, i.e the practice of cherry picking samples of data to achieve statistically significant results. This is often referred to as p-hacking.

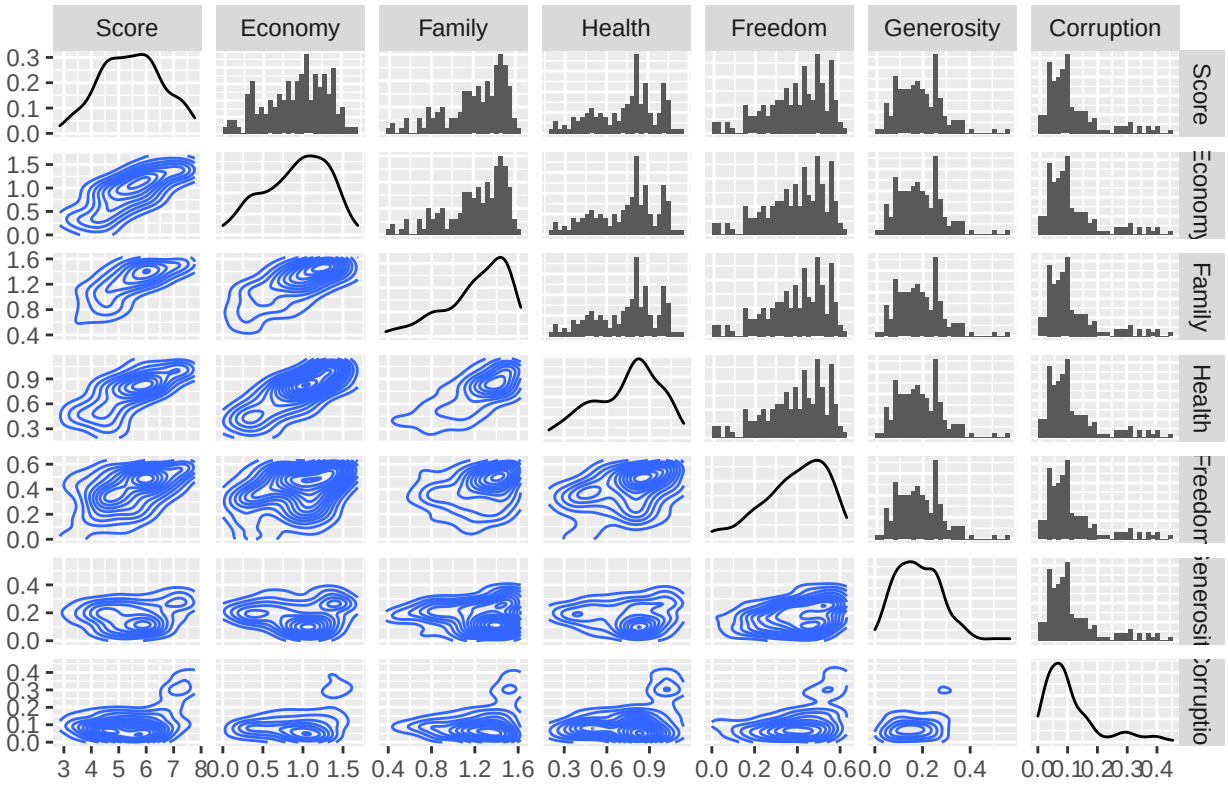
This latter case being excluded, we need to test if there is a case of **spurious correlation**. This maybe the case with the presence of outliers in the distributions, which we observed previously. We do so by using the **Sperman correlation**, i.e. by computing the correlation on the ranks of the values. The resulting Sperman correlation matrix, shown below, does not highlight any significant change in correlation among variables, therefore suggesting that spurious correlation can be excluded.



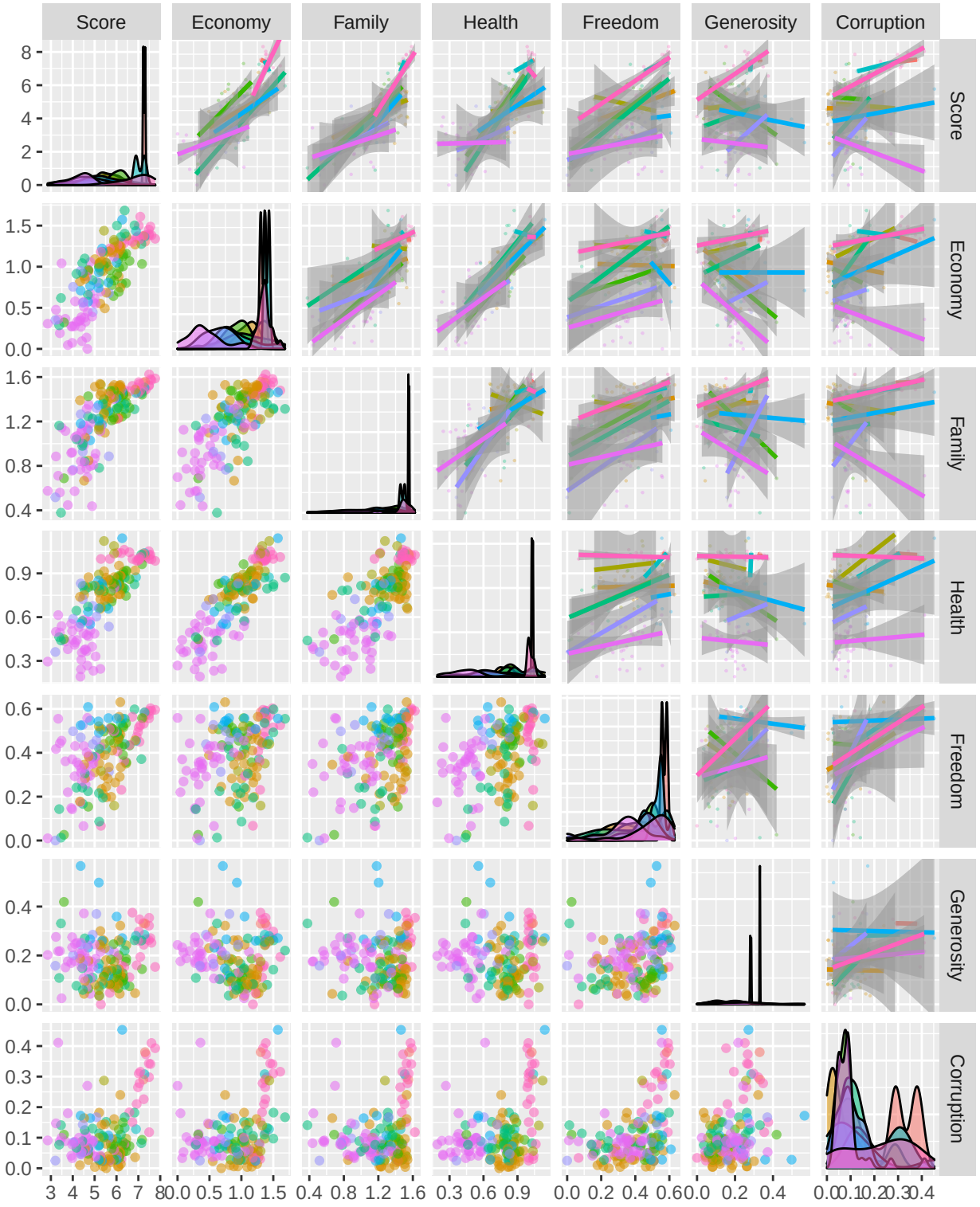
Confounding is another common reason that leads to associations begin misinterpreted: if X and Y are correlated, we call Z a confounder if changes in Z causes changes in both X and Y. In our specific situation, Economy could be a confounder of Heath and Happiness, and therefore explain the strong correlation among the two. "Region", a non-numerical variable that has not been studied in this section, could be a confounder of Economy and Health, and so forth. Unfortunately, we do not have enough data to be able to stratify and look at this phenomena analytically, and therefore we cannot exclude a-priori that some confounding exists.

The tables that follow are based on the ggpairs function under the *GGally* library, which is useful for exploring multiple distributions simultaneously. More insight is provided by displaying pair plots in a matrix format: the pairs are set among the six predictors, with different plots such as scatter or density plots.

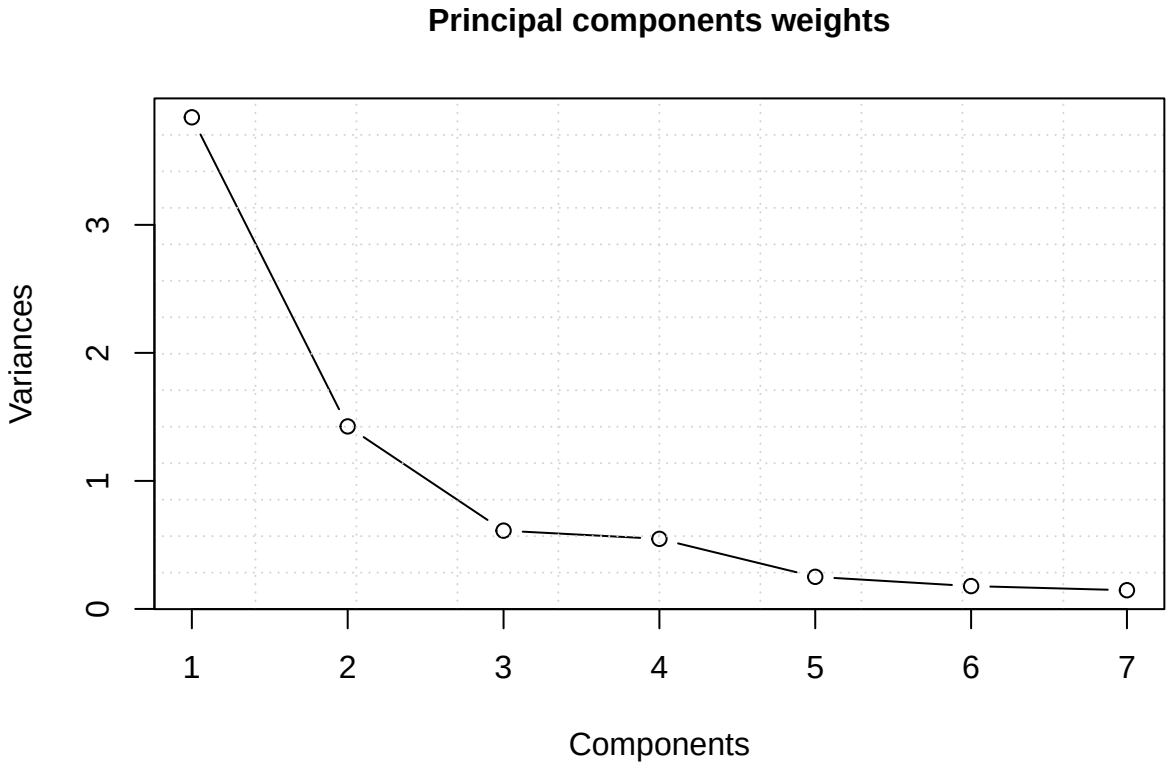




In this GGally plot, each color denotes a Region. One thing worth noting is that the scatter plots between happiness score and most variables have an approximately oval shape, they behave like **bivariate normal distributions**. This is apparent for variables such as Economy, Family, Health and Freedom, while it is less obvious for the last two variables, Generosity and Corruption. Overall, this seems to support the argument that linear regression (and linear models in general) may be an effective method to predict happiness.



Not surprisingly, the PCA analysis shows that two principal components alone explain more than half of the total variability. Three components explain the vast majority of it. Given the limited size of the database, in this current work we will not perform any dimension reduction analysis, but the PCA results confirm high causation particularly between 2-3 variables.

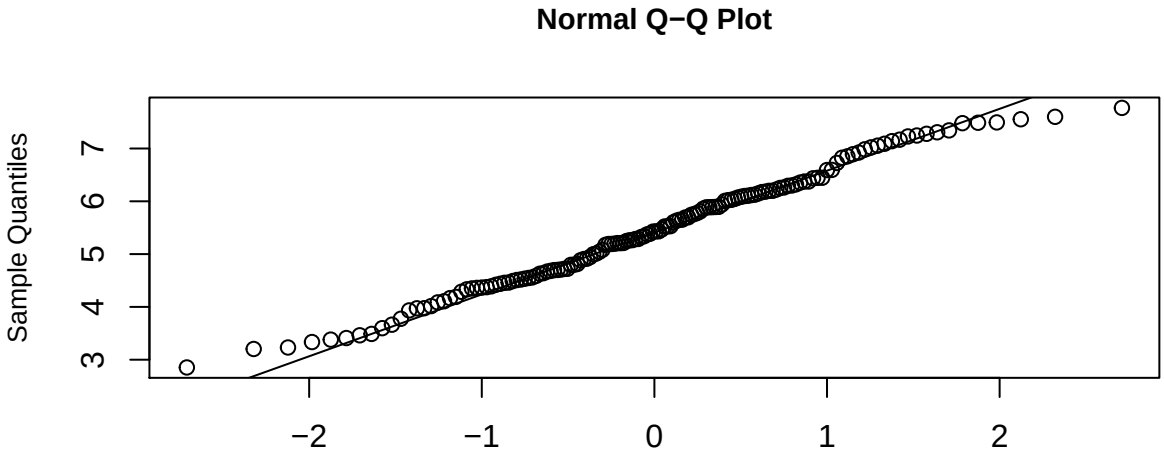


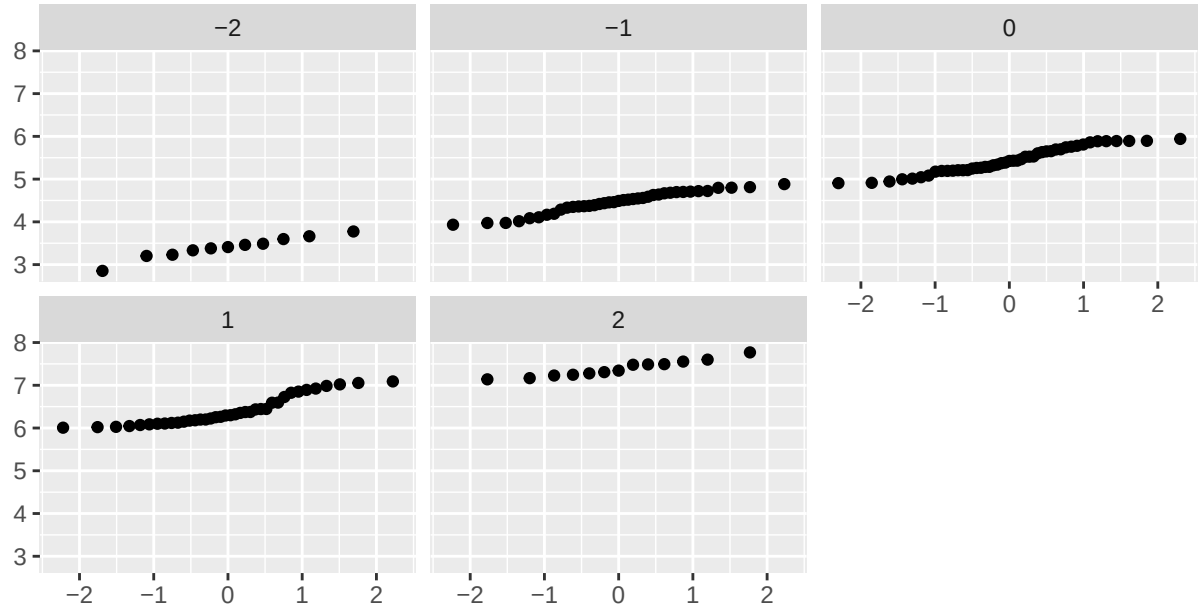

```
## Standard deviations (1, .., p=7):
## [1] 1.9596091 1.1940222 0.7820080 0.7398382 0.5007579 0.4223258 0.3823981
##
## Rotation (n x k) = (7 x 7):
##                    PC1        PC2         PC3          PC4         PC5
## Score      -0.47225166  0.0275685 -0.07667281 -0.006462619  0.17214438
## Economy    -0.45314393  0.2102228  0.07743931  0.245819384 -0.33324091
## Family     -0.43493443  0.2037749 -0.30820063  0.012766187  0.70481329
## Health     -0.45299921  0.1895337  0.06493141  0.234276898 -0.47547858
## Freedom    -0.33233112 -0.3595836 -0.19362865 -0.798434275 -0.25158190
## Generosity -0.04890026 -0.6998847 -0.51067984  0.489789714 -0.07616527
## Corruption -0.25382572 -0.5083898  0.76852766  0.084209083  0.25957765
##                    PC6         PC7
## Score      -0.84869594 -0.14275638
## Economy     0.05553424  0.75492343
## Family      0.42105963 -0.01021025
## Health      0.26123670 -0.63674363
## Freedom     0.13578530  0.05945744
## Generosity  0.02759925  0.02279127
## Corruption  0.10906861 -0.01048332
```

Linear Models

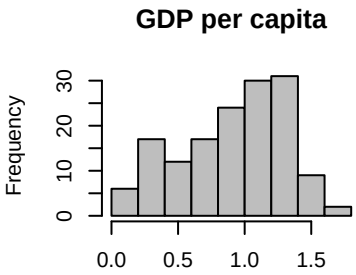
Data visualization allowed us to understand some of the features of the variables and of their distributions. One quality we noted is that most if not all variables and the happiness score seem to behave like bivariate normal distributions, hence supporting the argument that linear models may fit well to predict happiness. We also noticed strong correlations.

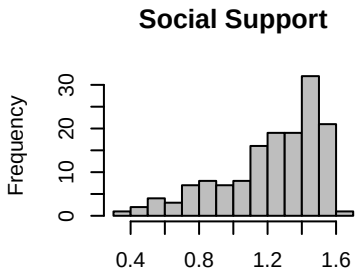
With the qq-plots below, we formalize this argument by comparing theoretical quantiles with sample quantiles (in this case, the entire population, given the small number of observations). The qq-plots do not provide a strong indication of non-normality: the plots are fairly straight across the different levels of standardized scores and overall. We observe there are some deviations along the tails. Next page, the histogram itself resembles fairly accurately a bell-shaped distribution, with perhaps a slight negative skewness (more data on the upper half) which is consistent with the tails of the qq-plots. The histograms of the variables, however, feature high skewness, both negative (Family, Life Exp.) and positive (Corruption, Generosity).

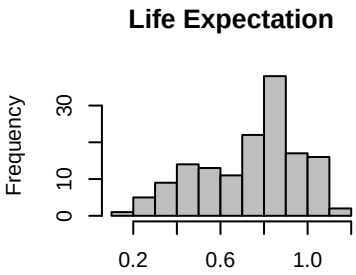


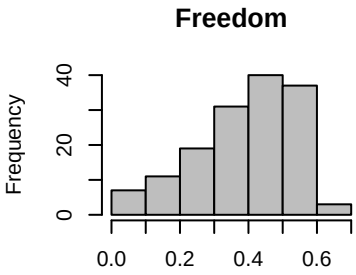


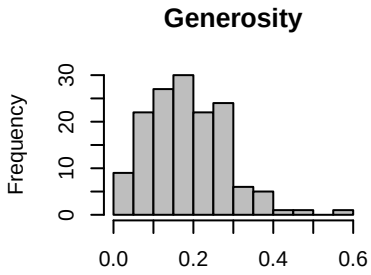

Happiness Score
Frequency
0
5
10
15
20
25
3
4
5
6
7
8


GDP per capita
Frequency
0
10
20
30
0.0
0.5
1.0
1.5


Social Support
Frequency
0
10
20
30
0.4
0.8
1.2
1.6


Life Expectation
Frequency
0
10
30
0.2
0.6
1.0


Freedom
Frequency
0
20
40
0.0
0.2
0.4
0.6


Generosity
Frequency
0
10
20
30
0.0
0.2
0.4
0.6

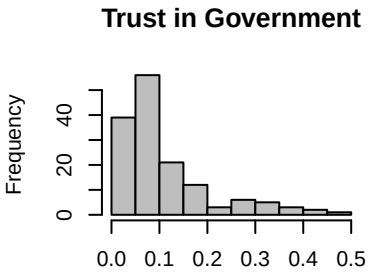

Trust in Government
Frequency
0
20
40
0.0
0.1
0.2
0.3
0.4
0.5

Table 1: Normality Tests

	p-value	skweness	kurtosis
Score	0.187888792541336	-0.000180676523365972	-0.65841225002281
Economy	0.00132177339118133	-0.380346227063542	-0.777791339072924
Family	8.55704620289585e-08	-0.91091317336391	0.0362301499195552
Health	9.07784806222984e-05	-0.486147153812309	-0.679453399293307
Freedom	4.41755669458474e-05	-0.681749653026499	-0.189285051942126
Generosity	0.000543585457904786	0.735022347928727	0.96726483252702
Corruption	2.52649353796356e-12	1.61367095523512	2.10869881126377

The Shapiro-Wilk normality test shows a **p-value of 0.188** for the Score distribution, and it is above 0.05 for most variables. For Economy and Generosity, we do not exceed the significant p-value level of 0.05, which would indicate rejection of the null hypothesis, i.e. rejection of the normality hypothesis. However, the sample population is very small and it is worth noting that such fit tests are more likely to "fail to reject" (type I error) when the sample size is small.

Additional tests performed are **skewness** and **kurtosis**. A normal distribution has skewness equal to 0: this test is met for the Score distribution, it is not met for the other variables. A normal distribution has kurtosis equal to 3: this test is NOT met. With negative excess kurtosis, we have platykurtic distributions with thinner tails and its central peak lower and broader, as observed in the histograms.

All considered, the deviations from normality seem acceptable and do not seem to materially impact our analysis and its results.

Linear Regression

With the previous results giving some comfort on the normality of the distributions, we proceed with linear regression. The approach is to find the values that minimize the distance of the fitted model to the data, i.e. we find those estimates that minimize the residual sum of squares (RSS):

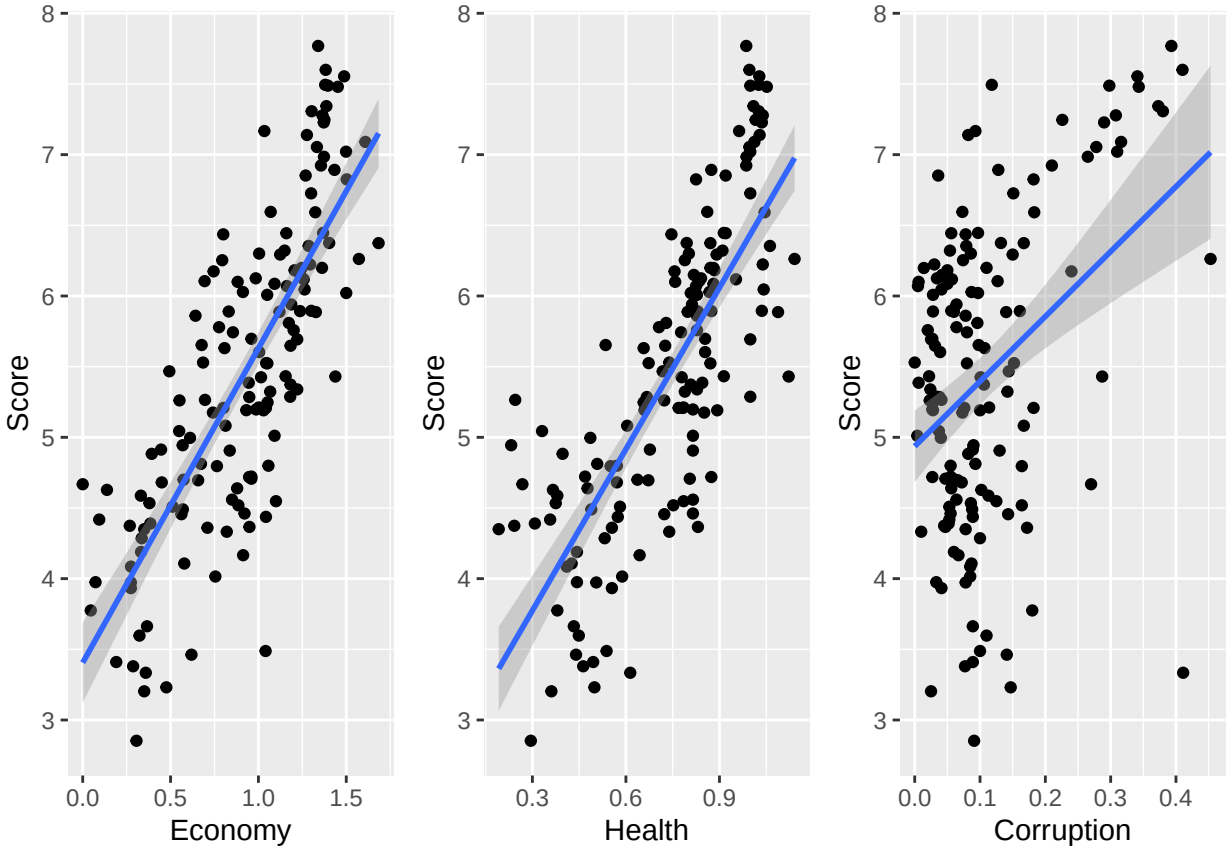
$$RSS = \sum_{i=1}^{n} \Big(y_i - (\beta_0 + \beta_1 x_i) \Big)^2$$

The tables below show the regression results using, for a start, one variable at once: Economy, Health and Corruption. Regression is fairly robust in the first two cases; less so in the third case where fitting Score against Corruption seems to achieve weaker results if we consider R-squared, p-value and residual standard error. We visualize this in the gg-plots with 95% confidence interval.

```
##
## Call:
## lm(formula = Score ~ Economy, data = dataset)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.23282 -0.47779  0.00788  0.50506  1.46175
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.4054     0.1440   23.65   <2e-16 ***
## Economy       2.2243     0.1443   15.41   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.685 on 146 degrees of freedom
## Multiple R-squared:  0.6193, Adjusted R-squared:  0.6167
## F-statistic: 237.5 on 1 and 146 DF,  p-value: < 2.2e-16
```

```
##
## Call:
## lm(formula = Score ~ Health, data = dataset)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.6356 -0.4709  0.0928  0.5484  1.7004
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.6318     0.1965   13.39   <2e-16 ***
## Health        3.8074     0.2542   14.98   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6972 on 146 degrees of freedom
## Multiple R-squared:  0.6058, Adjusted R-squared:  0.6031
## F-statistic: 224.3 on 1 and 146 DF,  p-value: < 2.2e-16
```

```
##
## Call:
## lm(formula = Score ~ Corruption, data = dataset)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.4903 -0.7569  0.1782  0.8267  2.0157
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   4.9363     0.1283  38.466  < 2e-16 ***
## Corruption    4.5938     0.8740   5.256 5.12e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.018 on 146 degrees of freedom
## Multiple R-squared:  0.1591, Adjusted R-squared:  0.1534
## F-statistic: 27.63 on 1 and 146 DF,  p-value: 5.119e-07
```

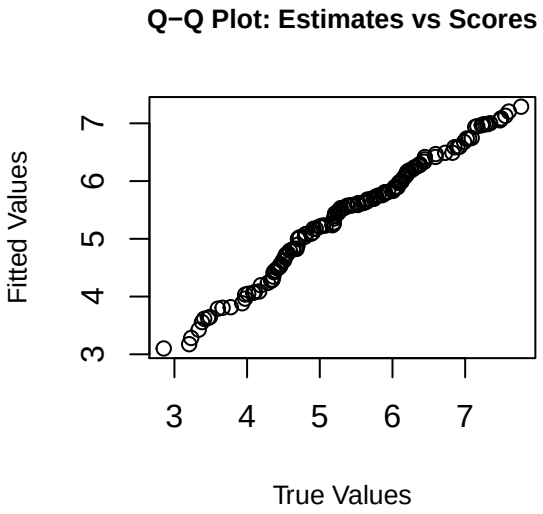


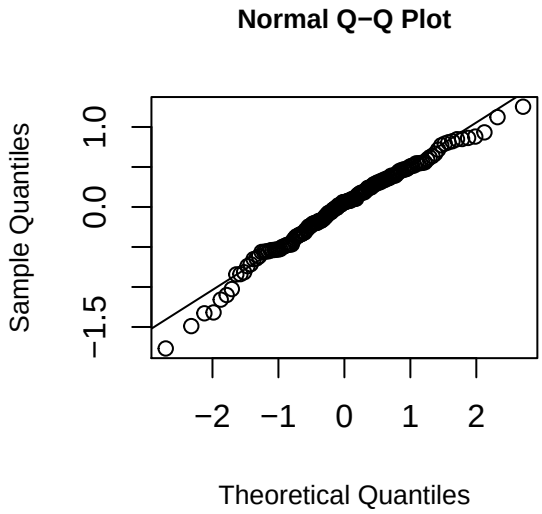
We now run the same regression, this time using all 6 predictors: a **multiple linear regression**. Results are more robust: R-squared is 77% and RSS is 0.54.

The two charts below represent: 1) a quantile-quantile (Q-Q) plot of the fitted values of our regression model against the actual happiness scores; 2) a Q-Q Norm plot of the residuals of our regression model. The estimated and actual distributions seem to have similar shape and location.

```
fit <- lm(Score ~ ., data = dataset[,5:11])
predictions <- predict(fit, se.fit = TRUE)
summary(fit)
```

```
##
## Call:
## lm(formula = Score ~ ., data = dataset[, 5:11])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.76885 -0.34341  0.06139  0.36241  1.25352
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.7340     0.2293   7.563 4.61e-12 ***
## Economy       0.7660     0.2347   3.264  0.00138 **
## Family        1.2447     0.2569   4.844 3.30e-06 ***
## Health        1.0081     0.3972   2.538  0.01224 *
## Freedom       1.4074     0.3840   3.665  0.00035 ***
## Generosity    0.4062     0.5118   0.794  0.42871
## Corruption    1.0727     0.5638   1.902  0.05915 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5408 on 141 degrees of freedom
## Multiple R-squared:  0.7709, Adjusted R-squared:  0.7611
## F-statistic: 79.06 on 6 and 141 DF,  p-value: < 2.2e-16
```



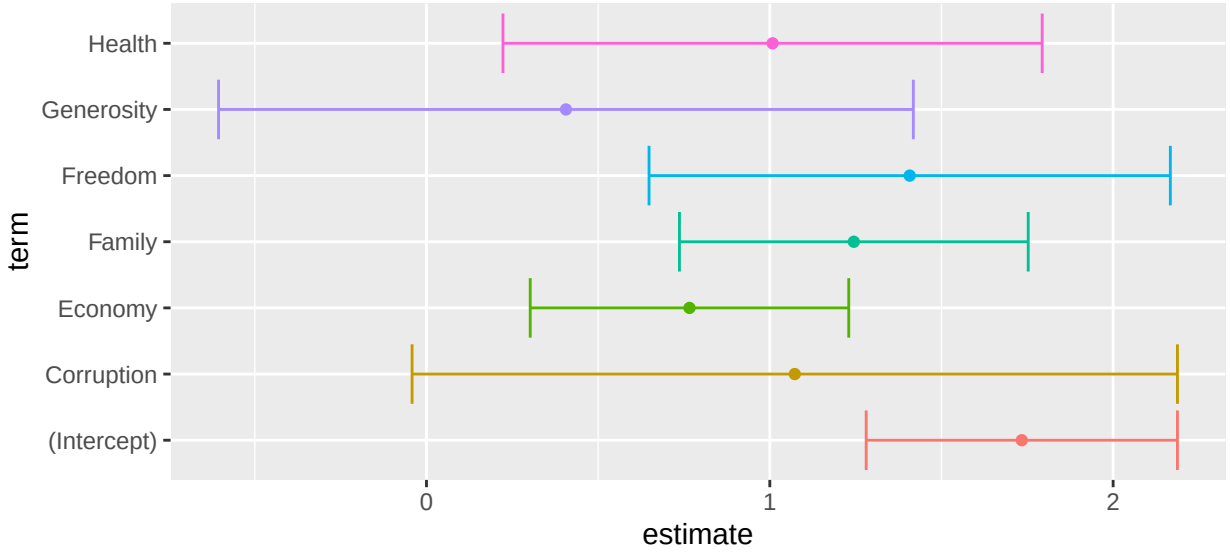


To complete our linear regression analysis, we use the *broom* package to present some further results using the functions *tidy, glance and augment.*

```
## # A tibble: 7 x 7
##   term        estimate std.error statistic  p.value conf.low conf.high
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>    <dbl>     <dbl>
## 1 (Intercept)    1.73      0.229     7.56  4.61e-12   1.28        2.19
## 2 Economy        0.766     0.235     3.26  1.38e- 3   0.302       1.23
## 3 Family         1.24      0.257     4.84  3.30e- 6   0.737       1.75
## 4 Health         1.01      0.397     2.54  1.22e- 2   0.223       1.79
## 5 Freedom        1.41      0.384     3.66  3.50e- 4   0.648       2.17
## 6 Generosity     0.406     0.512     0.794 4.29e- 1  -0.606       1.42
## 7 Corruption     1.07      0.564     1.90  5.91e- 2  -0.0420      2.19
```

```
## # A tibble: 1 x 11
##   r.squared adj.r.squared sigma statistic  p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>    <dbl> <int>  <dbl> <dbl> <dbl>
## 1     0.771         0.761 0.541      79.1 1.20e-42     7  -115.  247.  271.
## # ... with 2 more variables: deviance <dbl>, df.residual <int>
```

```
## # A tibble: 148 x 15
##    .rownames Score Economy Family Health Freedom Generosity Corruption .fitted
##    <chr>     <dbl>   <dbl>  <dbl>  <dbl>   <dbl>      <dbl>      <dbl>   <dbl>
##  1 1          7.77    1.34   1.59  0.986   0.596      0.153      0.393    7.05
##  2 2          7.6     1.38   1.57  0.996   0.592      0.252      0.41     7.13
##  3 3          7.55    1.49   1.58  1.03    0.603      0.271      0.341    7.20
##  4 4          7.49    1.38   1.62  1.03    0.591      0.354      0.118    6.95
##  5 5          7.49    1.40   1.52  0.999   0.557      0.322      0.298    6.94
##  6 6          7.48    1.45   1.53  1.05    0.572      0.263      0.343    7.09
##  7 7          7.34    1.39   1.49  1.01    0.574      0.267      0.373    6.98
##  8 8          7.31    1.30   1.56  1.03    0.585      0.33       0.38     7.07
##  9 9          7.28    1.36   1.50  1.04    0.584      0.285      0.308    6.97
## 10 10         7.25    1.38   1.48  1.02    0.532      0.244      0.226    6.74
## # ... with 138 more rows, and 6 more variables: .se.fit <dbl>, .resid <dbl>,
## #   .hat <dbl>, .sigma <dbl>, .cooksd <dbl>, .std.resid <dbl>
```



Machine Learning Models

In the following sections we apply machine learning to build, train and test a prediction model for the world happiness. We will consider different models: **linear and non-linear** models, as well as **supervised and unsupervised** models. We will discuss the merits of each model in the dedicated sections.

In machine learning, data comes in the form of: 1) the outcome we want to predict and, 2) the features that we use to predict the outcome. We want to build an algorithm that takes feature values as input and returns a prediction for the outcome when we don't know the outcome. The machine learning approach is to train an algorithm using a dataset for which we do know the outcome, and then apply this algorithm in the future to make a prediction when we don't know the outcome.

Before we proceed, we need to create our train and test sets. We will use the **train-set** to train the model (machine learning!) and fine-tuning parameters via **cross validation**. Once the models have been trained (or "fitted"), we will use them to make predictions and we will measure their performance against the true values of the **test-set**. This procedure will prevent any form of over-training or over-fitting.

We will use the *caret* package (short for "Classification And Regression Training") to efficiently train and fit our models. Caret package is a comprehensive framework for building machine learning models in R. We will measure prediction accuracy of each model in the form of **Residual Mean Squared Error ("RMSE")**:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$$

First thing in order, we partition our dataset using the caret package: the function *createDataPartition* generates indexes by randomly splitting the data into train and test sets. The train set is used to develop the algorithm and fit the parameters (we call this *supervised machine learning*); the test set is used to provide an unbiased evaluation of a fully-trained model.

To split the dataset between train and test sets, there are two competing concerns we need to handle: with less training data, parameter estimates have greater variance; with less testing data, performance statistics have greater variance. Given our database has a very small number of observations (148 Countries), cross-validation becomes more important and therefore we should devote a larger proportion of data to train and fit the models. We use 80% of observations for training purposes and the remaining 20% for testing. Additionally, we create the RMSE function to evaluate the different models.

```
# Function that computes the residual means squared error

RMSE <- function(true_ratings, predicted_ratings){
    sqrt(mean((true_ratings - predicted_ratings)^2))
}


# Data partitioning into train-set and test-set

set.seed(1979, sample.kind="Rounding")

index <- createDataPartition(y = dataset$Score, times = 1, p = 0.2, list = FALSE)
train_set <- dataset[-index, 5:11]
test_set <- dataset[index, 5:11]
```

```
##                 Train Set Test Set
## N. Features             7        7
## N. Observations       116       32
```

Conditional probabilities and expectations. Smoothing techniques.

In machine learning applications, we rarely can predict outcomes perfectly. The most common reason for not being able to build perfect algorithms is that it is impossible. To see this, note that most datasets will include the same exact observed values for all predictors, but with different outcomes. In a machine learning model however, equal inputs (the predictors) implies equal outputs (the predictions). To overcome this issue, we make use of probabilistic representations of the problem. Outcomes with the same observed values for the predictors may not all be the same, but we can assume that they all have the same probability of this class or that class. This is **conditional probability**.

These probabilities guide the construction of an algorithm that makes the best prediction: for any given x, we will predict the class k with the largest probability. We refer to this as Bayes' Rule or conditional expectation. Note that the expected value has an attractive mathematical property: it minimizes the MSE. And this is the reason we are using RMSE as a measure of accuracy of our prediction models. We have now established the connection between conditional probabilities and machine learning.

Discriminative models calculate the conditional probability $p(Y|X = x)$ directly and do not consider the distribution of the predictors. They discriminate one class (or object) from another: a digit, an animal, the sex of a person. For this reason they are most often used for classification purposes. However, knowing the distribution of the predictors may be useful. Methods that model the joint probability/distribution $p(X, Y)$ are referred to as **generative models**: they model how the entire data, X and Y, are generated. The theoretical foundation is provided by Bayes'theorem. Naive Bayes or Quadratic Discriminant Analysis (QDA) are two examples of generative models. Discriminative training algorithms tend to be more complex albeit they look simpler. The intuition behind this is that they are trying to learn something more subtle. They make however fewer assumptions about the underlying data distribution and rely more on data quality. With many classes of predictors, and when it is not clear which class is more discriminative, generative models may be adopted. This is not the case here, therefore we will use discriminative methods, which are also computationally less onerous.

We now clarify how these concepts are tackled from a numerical perspective, i.e. we introduce the concept of **smoothing** (or "curve fitting"), a powerful technique used across data analysis. The concepts behind smoothing techniques are extremely useful in machine learning because conditional expectations / probabilities can be thought of as trends of unknown shapes that we need to estimate in the presence of uncertainty. The general idea of smoothing is to group data points into strata in which the value of *f(x)* can be assumed to be constant. We can make this assumption because we think *f(x)* changes slowly and, as a result, *f(x)* is almost constant in small windows of time. The idea behind **bin smoothing** is to make this calculation with each value of x as the center. By computing this mean for every point, we form an estimate of the underlying curve *f(x)*. The final result from the bin smoother is normally quite wiggly. One reason for this is that each time the window moves, two points change. We can attenuate this somewhat by taking weighted averages that give the center point more weight than far away points, with the two points at the edges receiving very little weight. This technique is called **kernel**; the ksmooth function provides a "smoother" option which uses the normal density to assign weights. A further improvement can be achieved by using **local weighted regression (loess)**: using Taylor's Theorem, instead of assuming the function is approximately constant in a window, we assume the function is locally linear. Clearly, how we set the smoothing parameters is critical in machine learning.

There are many more sophisticated techniques that the literature provides. With the key concepts clarified, we are now ready to build a machine learning model.

Model 1: Generalized Linear Model (GLM)

We start with a generalized linear model, which is a flexible generalization of ordinary linear regression. It is worth recalling that, if the pair (X, Y) follow a bivariate normal distribution, then the conditional expectation (what we want to estimate) is equivalent to the regression line.

GLM is a more flexible linear predictive model in that it allows for variables that have distribution models other than a normal distribution. Ordinary linear regression predicts the expected value of an unknown quantity as a linear combination of a set of observed values (predictors): a constant change in a predictor leads to a constant change in the variable we want to predict. This is appropriate when the latter has a normal distribution, but not for many other arbitrary distributions that we observe in nature (exponential, log-linear, binomial, Bernoulli etc.). Generalized linear models cover all these situations by allowing the unknown variable to have arbitrary distributions (rather than simply normal distributions). It is a way of unifying various other statistical models, including linear regression, logistic regression and Poisson regression. As we will note further on, we may wish to drop linearity to achieve higher accuracy in our predictions.

Below we show the code for fitting the model and to make predictions. Note that there is no tuning parameter for this model. The fitting parameters, predictions and results are below. The **RMSE** for the GLM model is **0.589**. This compares with a RSS of the linear regression model 0.541. However, note that the linear regression was performed on the entire population, i.e. we used our knowledge of the true results of the Happiness Score and computed the best fitting line: it's a standard regression where we estimated the parameters. In this machine learning exercise, we made predictions of the Happiness Score pretending we do not know the true outcome when we train the model.

```
glm_fit <- train(Score ~ ., method = "glm", data = train_set)
glm_predictions <- predict(glm_fit, test_set, type = "raw")

glm_fit
```

```
## Generalized Linear Model
##
## 116 samples
##   6 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 116, 116, 116, 116, 116, 116, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.5715286  0.7664329  0.4516386
```

```
summary(glm_predictions)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.897   5.149   5.533   5.457   5.746   7.226
```

```
glm_rmse <- RMSE(test_set$Score, glm_predictions)
glm_rmse
```

```
## [1] 0.5894279
```

Model 2: k-nearest Neighbors (kNN)

With k-nearest neighbors (kNN) we estimate the conditional probability of two points $p(x_1, x_2)$ in a similar way to standard bin smoothing. However, kNN is easier to adapt to multiple dimensions and therefore is suited to our case. First we define the distance between all observations based on the features. Then, for any point (x_1, x_2) for which we want an estimate of $p(x_1, x_2)$, we look for the k nearest points to (x_1, x_2) and then take an average of the 0s and 1s associated with these points. We refer to the set of points used to compute the average as the *neighborhood*.

K-fold cross validation. We can control the flexibility of our estimate, in this case through the k parameter: larger k_s result in smoother estimates, while smaller k_s result in more flexible and more wiggly estimates. The *caret* package provides a function that performs cross validation for us. By default, cross validation is performed by taking 25 bootstrap samples comprised of 25% of the observations. For the kNN method, the default is to try k = 5, 7, 9.

The predict function for knn produces a probability for each class. This algorithm generates an **RMSE** of **0.507**, a significant improvement from the GLM performance.

```
knn_fit <- train(Score ~ ., method = "knn", data = train_set)
knn_predictions <- predict(knn_fit, test_set, type = "raw")

knn_fit
```

```
## k-Nearest Neighbors
##
## 116 samples
##   6 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 116, 116, 116, 116, 116, 116, ...
## Resampling results across tuning parameters:
##
##   k  RMSE       Rsquared   MAE
##   5  0.5460347  0.7854470  0.4255990
##   7  0.5305195  0.7969088  0.4102972
##   9  0.5268705  0.8007305  0.4096738
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 9.
```

```
summary(knn_predictions)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.951   4.916   5.454   5.401   5.797   7.422
```

```
knn_rmse <- RMSE(test_set$Score,knn_predictions)
knn_rmse
```

```
## [1] 0.5072762
```

Model 3: Local Weighted Regression (loess)

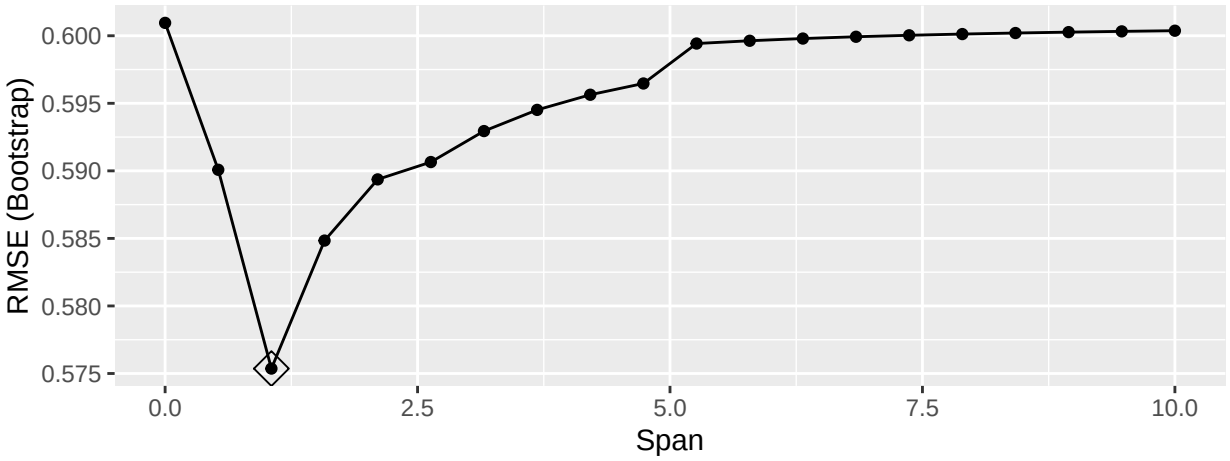
As noted previously, standard bin smoothing or kNN smoothing assume that a function is (approximately) constant within the window, so we need (very) small windows for this assumption to be true. kNN for instance is a type of *lazy learning*, where the function is only approximated locally, and all computation is deferred until function evaluation. The consequence of this is that we end up with a small number of data points to average and this could results in a imprecise estimate (the boundary of the curve we want to fit ends up somewhat wiggly).

A local weighted regression (loess) permits us to consider larger window sizes (we call these *span*): instead of assuming that the function is approximately constant in a window, we assume the function is locally linear. This is a more flexible method to fit a curve.

We perform cross-validation of the parameter *span* by using the *tuneGrid* parameter. The best fitting parameter and results are below. **RMSE** under the loess model is **0.559**, higher than in the case of knn.

```
grid <- expand.grid(span = seq(0, 10, len = 20), degree = 1)
train_loess <- train(Score ~ ., method = "gamLoess",
                     tuneGrid=grid,
                     data = train_set )

ggplot(train_loess, highlight = TRUE)
```




```
train_loess$bestTune
```

```
##       span degree
## 3 1.052632      1
```

```
loess_fit <- train(Score ~ ., method = "gamLoess",
                   span = train_loess$bestTune,
                   degree = 1,
                   data = train_set)
loess_fit
```

```
## Generalized Additive Model using LOESS
##
## 116 samples
```

```
##   6 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 116, 116, 116, 116, 116, 116, ...
## Resampling results:
##
##   RMSE       Rsquared  MAE
##   0.5708878  0.759956  0.440272
##
## Tuning parameter 'span' was held constant at a value of 0.5
## Tuning
##  parameter 'degree' was held constant at a value of 1
```

```
loess_predictions <- predict(loess_fit, test_set)
summary(loess_predictions)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.734   5.061   5.447   5.371   5.653   7.338
```

```
loess_rmse <- RMSE(test_set$Score,loess_predictions)
loess_rmse
```

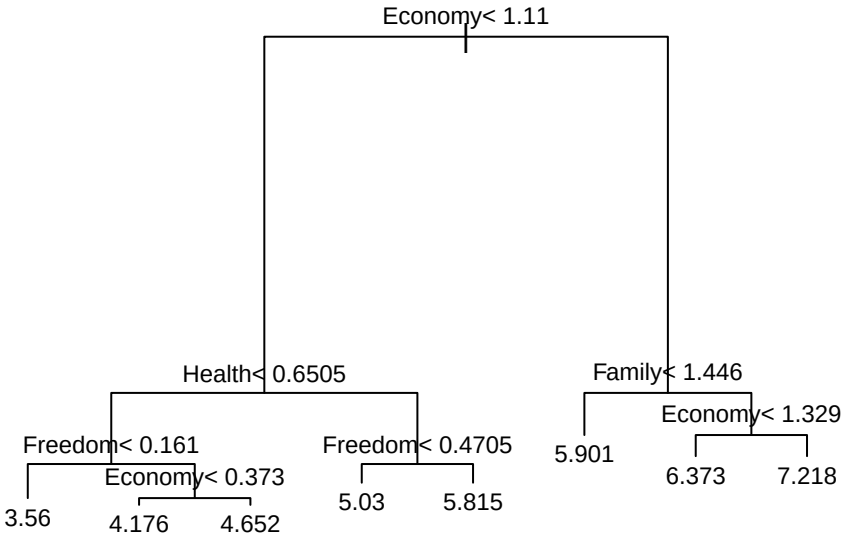
```
## [1] 0.5589091
```

Model 4: Regression Tree

A tree is basically a flow chart of yes or no questions. The general idea is to define an algorithm that uses data to create these trees with predictions at the ends, referred to as *nodes*. Decision trees operate by predicting an outcome Y by partitioning the predictors. When the outcome is continuous, we call the method a regression tree.

Regression trees create partitions recursively. We start the algorithm with one partition, the entire predictor space. After the first step we will have two partitions. After the second step we will split one of these partitions into two and will have three partitions, then four, five, and so on.

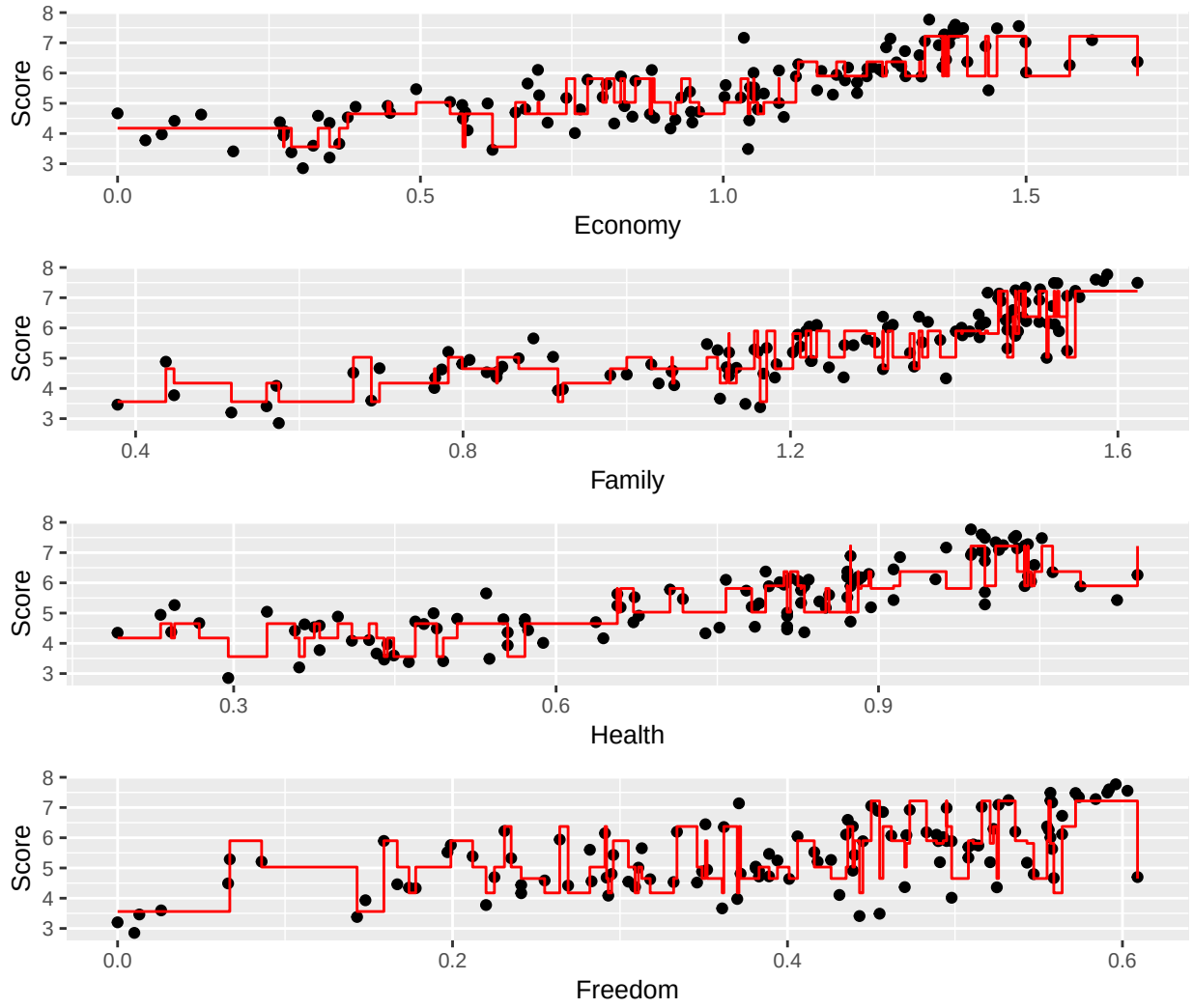
The chart below describes how the nodes are built in our case: we start with predictor Economy, and we decide to split it at a value of 1.11. One of these regions uses Health (a second predictor) and is further split at a value 0.6505 in two partitions. The other branch uses a third predictor, Family, and recursively we split this region in two more branches and so on.



The algorithm stopped partitioning at 7, with 8 nodes. Note the *complexity parameter*. Every time we split and define two new partitions, our train-set RSS decreases. This is because with more partitions, our model has more flexibility to adapt to the training data. In fact, if you split until every point is its own partition, then RSS goes all the way down to 0, since the average of one value is that same value. To avoid this, the algorithm sets a minimum for how much the RSS must improve for another partition to be added. This is the meaning of *cp*: the RSS must improve by a factor of *cp* for the new partition to be added.

By default, *cp* is set at 0.01. Large values of *cp* will therefore force the algorithm to stop earlier which result in less nodes. The other parameter we control is the minimum number of observations required in a partition. By default, 20 observations: *minsplit*=20. If we set $cp = 0$ and $minsplit = 2$, then our prediction is as flexible as possible and our predictor is our original data. The computational effort is however large. In addition, we will likely incur in over-training. The larger these values are the more data is averaged to compute a predictor and thus reduce variability. The drawback is that it restricts flexibility. As usual, the way to optimize these parameters is by using cross-validation.

```
##           CP nsplit rel error    xerror       xstd
## 1 0.56806335      0 1.0000000 1.0116021 0.10651654
## 2 0.11341728      1 0.4319366 0.5641361 0.06863363
## 3 0.06722649      2 0.3185194 0.3627780 0.04265856
## 4 0.03367895      3 0.2512929 0.3416371 0.04405853
## 5 0.03264501      4 0.2176139 0.3766370 0.04581432
## 6 0.02981556      5 0.1849689 0.3639827 0.04621806
## 7 0.01079696      6 0.1551534 0.3376094 0.04579505
## 8 0.01000000      7 0.1443564 0.3207519 0.04226593
```



The code further below shows the model fitting and the results of the regression tree algorithm . **RMSE** is equal to **0.73**. This model is somewhat less accurate than the previous models. We will try to improve accuracy moving from a "tree" to a "forest". The random forest algorithm is discussed next.

```
tree_fit <- rpart(Score ~ ., data = train_set)
tree_predictions <- predict(tree_fit, test_set)
summary(tree_predictions)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.560   4.935   5.030   5.263   5.815   6.373
```

```
tree_rmse <- RMSE(test_set$Score,tree_predictions)
tree_rmse
```

```
## [1] 0.7299938
```

Model 5: Random Forest

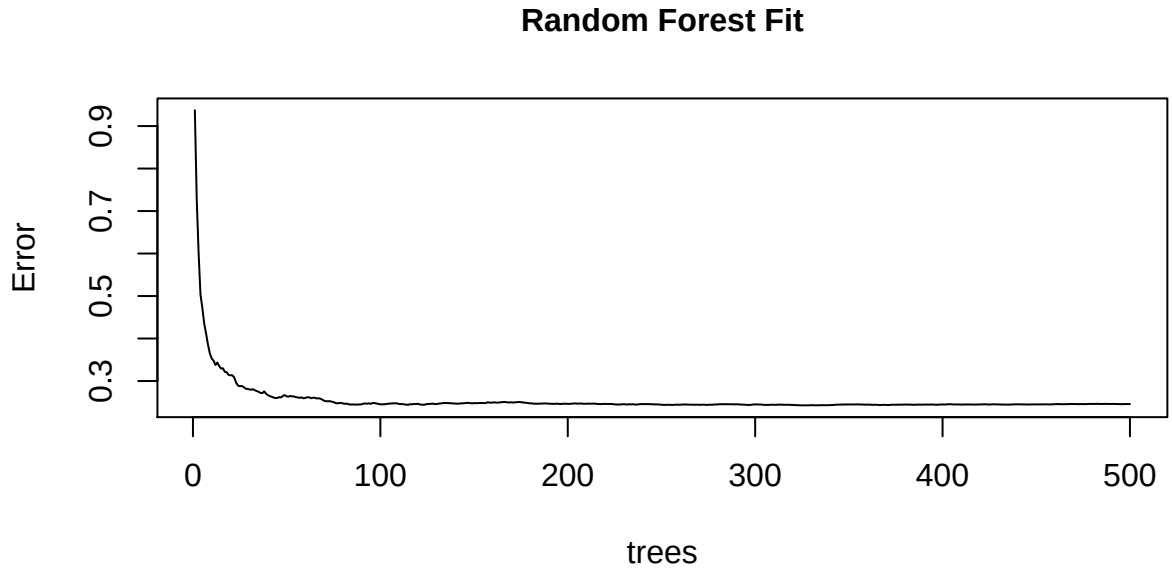
Random forests are a very popular machine learning approach that addresses some of the shortcomings of decision trees. The idea is to improve prediction performance and reduce instability by averaging multiple decision trees (a forest of trees constructed with randomness).

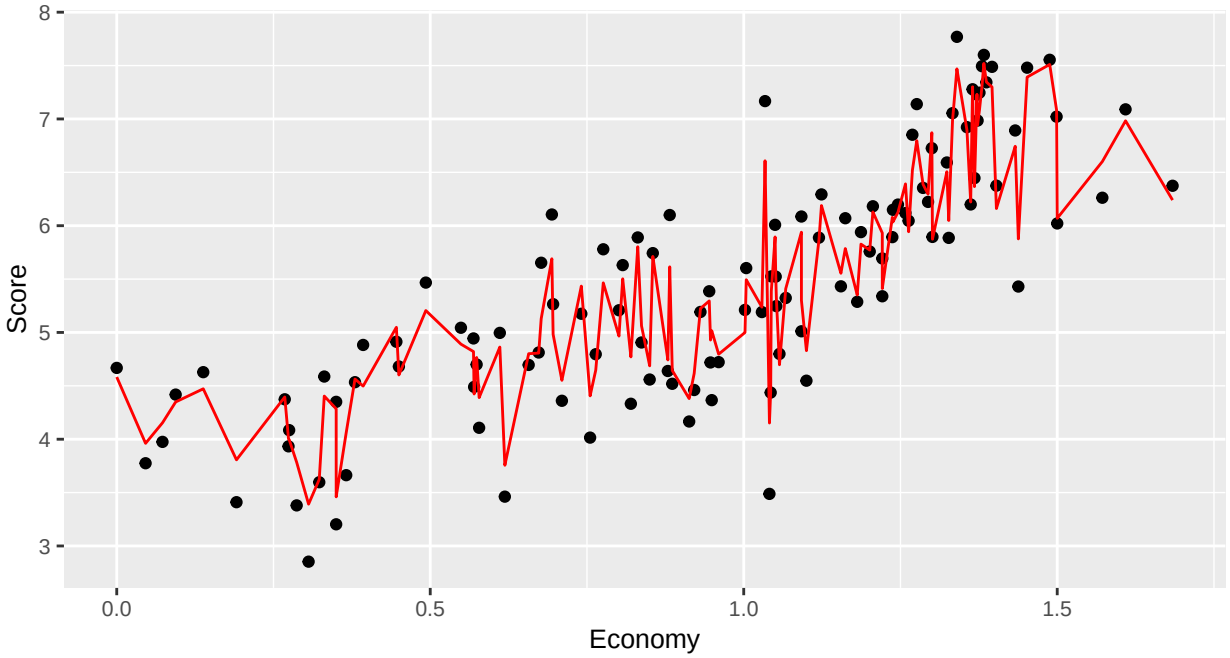
The first step is *bootstrap aggregation*: the idea is to generate many predictors, each using regression or classification trees, and then forming a final prediction based on the average prediction of all these trees. To assure that the individual trees are not the same, we use the bootstrap to induce randomness. These two features explain the name: the bootstrap makes the individual trees randomly different, and the combination of trees is the forest.

We obtain different decision trees from a single training set by applying two strategies: 1) we create a bootstrap training set by sampling N observations from the training set with replacement. This is the first way to induce randomness. 2) The second way random forests induce randomness is by randomly selecting features to be included in the building of each tree. A different random subset is selected for each tree. This reduces correlation between trees in the forest, thereby improving prediction accuracy. Note that often, many features can be informative but including them all in the model may result in overfitting.

In the chart below we see how the error rate of the algorithm drops as we add more trees. Effectively, accuracy improves significantly until we add about 50 trees, after which it stabilizes. From the chart further next we appreciate how a random forest is smoother than a regression tree. It is so because the average of many step functions can be smooth.

```
##
## Call:
##  randomForest(formula = Score ~ ., data = train_set)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##           Mean of squared residuals: 0.2457441
##                     % Var explained: 80.88
```





Below are the results of our model, which as expected marks a significant improvement when compared to the regression tree. The random forest **RMSE** is **0.578**.

Note however that this higher performance comes at the cost of losing interpretability. One way to alleviate this problem is to examine variable importance: we count how often a predictor is used in the individual trees. The caret package includes the function *varImp* that extracts variable importance from any model in which the calculation is implemented. As per below.

```
rf_predictions <- predict(rf_fit, test_set)
summary(rf_predictions)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.976   4.998   5.266   5.311   5.647   7.292
```

```
rf_rmse <- RMSE(test_set$Score,rf_predictions)
rf_rmse
```

```
## [1] 0.578062
```

```
varImp(rf_fit)
```

```
##              Overall
## Economy    41.021231
## Family     35.340061
## Health     34.633652
## Freedom    15.961690
## Generosity  5.201284
## Corruption 11.129018
```

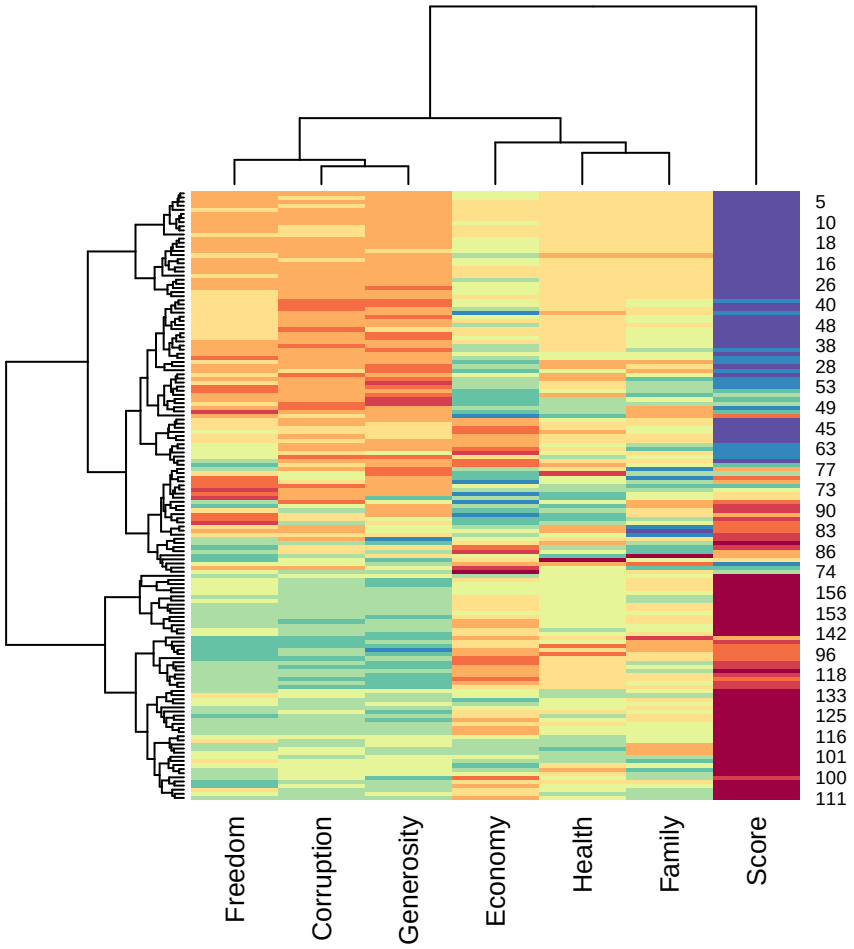
Model 6: Clustering - Heatmap

The algorithms used up to now are examples of a general approach referred to as supervised machine learning: we use the outcomes in a training set to supervise the creation of our prediction algorithm. There is another subset of machine learning referred to as unsupervised: in this case, we do not necessarily know the outcomes and instead are interested in discovering groups. These algorithms are also referred to as **clustering algorithm** since predictors are used to define clusters. There are applications in which unsupervised learning can be a powerful technique, in particular as an exploratory tool.

A first step in any clustering algorithm is defining a *distance* between observations or groups of observations. Then we need to decide how to join observations into clusters. There are many algorithms for doing this, we choose to use **Heatmap**. Other popular algorithms in this family are hierarchical clustering and k-means.

Heatmaps are a powerful visualization tool to discover clusters or patterns in your data. The idea is to plot an image of your data matrix with colors used as the visual cue and both the columns and rows ordered according to the results of clustering algorithm.

This is the result. The distance measures how different two data points are. It is calculated using the *dist()* function, whose own default is euclidean distance, and is already embedded into *heatmap()*. Note that it only measures the absolute distance between the points in space, and quite importantly, pays no attention to the "shape" of the curve. This makes it difficult to interpret.



However, if we consider the Pearson's correlation as a measure of similarity (with cor=1 meaning the shape of two distributions is identical, cor=-1 the exact opposite), then for clustering we can subtract the correlation matrix from 1 to get a measure of distance. We can do that for 10 observations, for example, and run a

naive cluster analysis, as shown below. It now makes more sense: the distance between one object and itself is zero. The two observations with the least distance (pair 3,6) will be clustered together, then recursively the next (pair 1,2), the next again (9,10) and so on. The heatmap is visualizing this process, only it uses color. This method is good as a visualization technique only, it will not generate an RMSE.

```
##               1           2           3           4           5           6
## 1  0.0000000000 0.0008916067 0.0048616962 0.018803871 0.006346040 0.0052307906
## 2  0.0008916067 0.0000000000 0.0027772926 0.016525580 0.003190025 0.0029302400
## 3  0.0048616962 0.0027772926 0.0000000000 0.010630219 0.001791625 0.0006660866
## 4  0.0188038709 0.0165255804 0.0106302187 0.000000000 0.007971411 0.0127539542
## 5  0.0063460405 0.0031900253 0.0017916253 0.007971411 0.000000000 0.0017905544
## 6  0.0052307906 0.0029302400 0.0006660866 0.012753954 0.001790554 0.0000000000
## 7  0.0035596306 0.0014013377 0.0017391057 0.016680967 0.002146484 0.0008274309
## 8  0.0026465273 0.0014191123 0.0050106028 0.013304786 0.002631264 0.0044723089
## 9  0.0046565064 0.0028157770 0.0013929561 0.009582112 0.001051625 0.0006789651
## 10 0.0088284286 0.0063291463 0.0016102651 0.007615497 0.002111312 0.0011738699
##               7           8           9          10
## 1  0.0035596306 0.002646527 0.0046565064 0.008828429
## 2  0.0014013377 0.001419112 0.0028157770 0.006329146
## 3  0.0017391057 0.005010603 0.0013929561 0.001610265
## 4  0.0166809675 0.013304786 0.0095821121 0.007615497
## 5  0.0021464837 0.002631264 0.0010516246 0.002111312
## 6  0.0008274309 0.004472309 0.0006789651 0.001173870
## 7  0.0000000000 0.003009572 0.0013185978 0.003545136
## 8  0.0030095715 0.000000000 0.0028498163 0.006675022
## 9  0.0013185978 0.002849816 0.0000000000 0.001034000
## 10 0.0035451359 0.006675022 0.0010339998 0.000000000
```

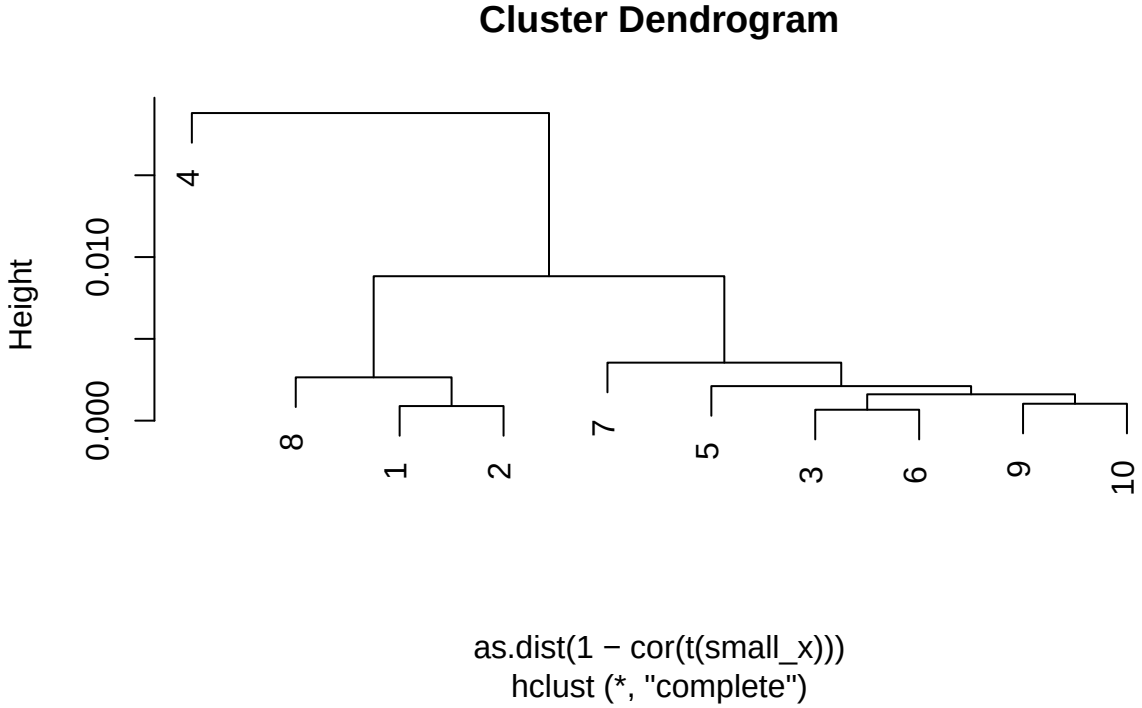
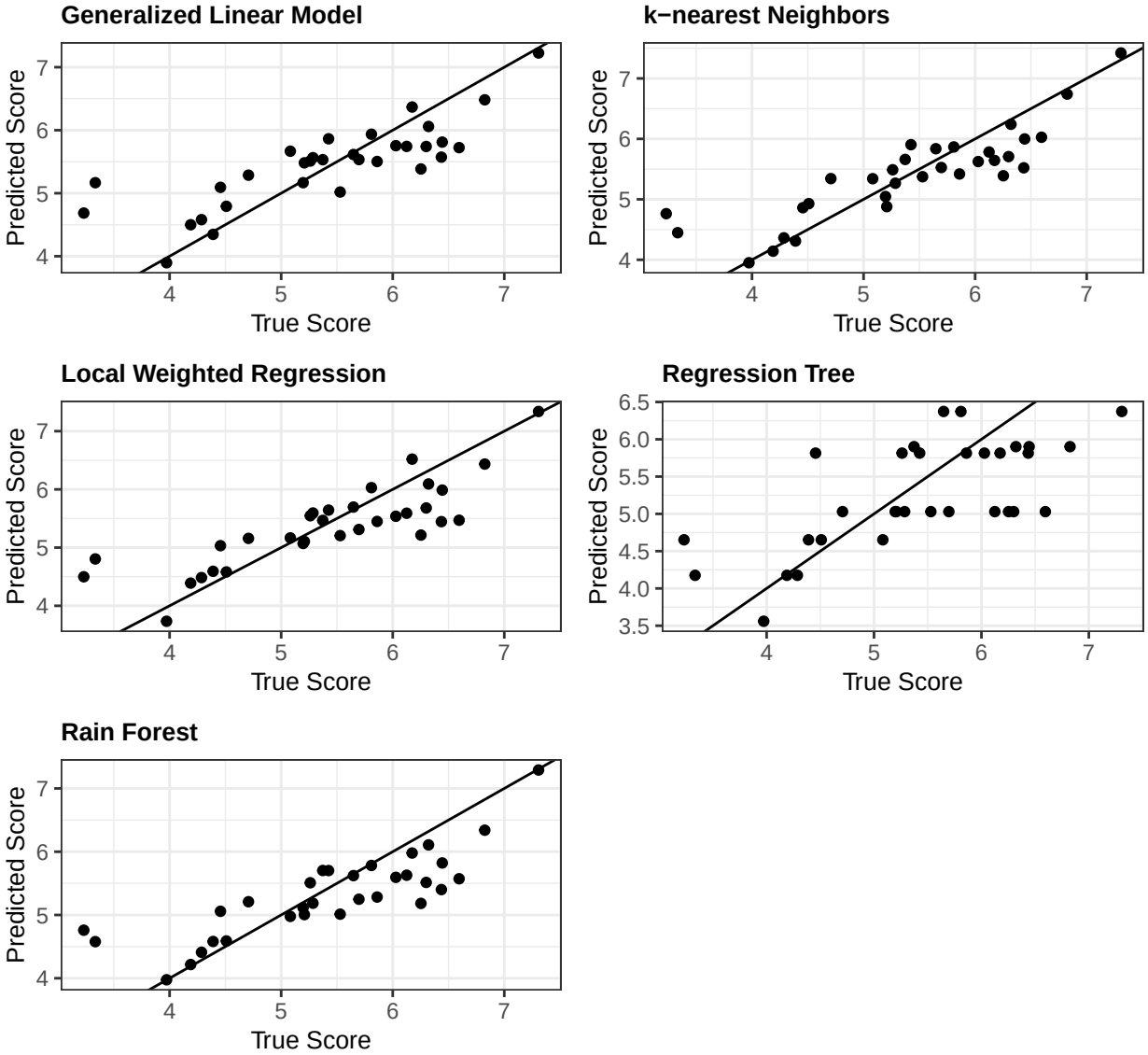


Table 2: Machine Learning Models - Predictive Accuracy (RMSE)

	RMSE	Linearity	Discriminative	Cross-Validated	Supervised
Generalized Linear Model	0.58942793379886	linear (parametric)	yes	no	yes
K-nearest Neighbors	0.507276166201701	locally constant (non-parametric)	yes	yes	yes
Local Weighted Regression	0.558909066621064	locally linear (non-parametric)	yes	yes	yes
Regression Tree	0.729993807673267	non-linear	yes	yes	yes
Random Forest	0.578062008685178	non-linear	yes	yes	yes
Heatmap	N/A	non-linear	yes	no	clustering

Results

The table on top of this page and the gg-plots below provide a summary of the algorithms performance, measured in the form of RMSE. The best performing model is kNN, with an RMSE of 0.507, followed by loess. GLM and random forest are not far behind, while regression tree is significantly less accurate in predicting the happiness score.



Conclusions

In this work, we have tried to analyze the data underlying the popular Gallup's poll on world happiness.

After some data exploration and some elementary statistical analysis, we have attempted to forecast the happiness score using machine learning as a predictive tool. We have applied different algorithms and for each model, measured the predictive accuracy in the form of *residual mean square errors*. We have used linear and non-linear models as well as supervised and un-supervised models. We have not used logistic regression as this applies to categorical data traditionally limited to only two-class classification problems (for example binary data such as 0 and 1, TRUE and FALSE, Male and Female). It is worth however noting that logistic regression is a specific case of a set of generalized linear models (GLM) that we have used. Similarly, we have not used generative classification algorithms such as Linear (LDA) or Quadratic Discriminant Analysis (QDA).

After deploying GLM regression, we have gradually relaxed our assumptions around the distribution of the observed variables, and have introduced more flexible models such as kNN and loess algorithms. These two models have performed best. Finally, we experimented with regression tree and random forest, and concluded with some clustering analysis in the form of a heatmap visualization.

Future Developments

As summarized in the previous section, we have reached satisfactory predictive results with some of our algorithms. All models scored an RMSE in the 0.5-0.6 region, with the exception of the regression tree (RMSE > 0.70). To improve our machine learning performance, we foresee future developments along two main directions: new predictors and new methods.

Predictors. Future work may try to establish the link between country happiness and some non-numerical (qualitative) variables. We think that the following features are worth exploring:
- *Geography*. There seems to be a divide between north and south hemisphere, this is largely measured by wealth and income. It may be worth understanding if there is causation with happiness. Is there a divide also between West and East?
- *Form of government*. Democracies around the world have provided for more equality among citizens, although in the recent past some of the largest democracies in the western world have struggled to deliver results, while some more absolutist regimes such as China have recorded years of growth and prosperity to an ever larger share of the population. Does a country governance affect happiness by simply impacting on one of the key variables we studied (freedom, economy, corruption), or there is more than a confounding effect?
- *Religion*. Bhutan, the original sponsor of the World Happiness Report, is a poor country under many measurable economic standards, yet with a reputation of having a happy, balanced society. Bhutan is a Buddhist country. Does Buddhism do a better job than other religions in "teaching" a community how to be happy?
- *Culture*. An individualistic culture is a society characterized by individualism, which is the prioritization or emphasis of the individual over the entire group, as opposed to collectivism. Which cultural model does a better job in achieving happiness?

Methodology. Across this study, we have used discriminative models. However, generative models may outperform discriminative models on smaller datasets if their assumptions place some structure that prevents overfitting. Generative models can learn the underlying structure of the data if the model assumptions are chosen correctly. On the contrary, discriminative algorithms are less tied to a particular structure, and in a complex world, assumptions are rarely perfectly satisfied. There is a trade-off that is worth exploring, and future development could look into deploying generative models to identify specific structures in the data that improves accuracy. This could also be supported by some more robust clustering analysis, which in this current work we have touched upon only on the surface.

Given the small size of the dataset, we do not foresee, instead, the advantage of pursuing dimension reduction techniques, such as PCA and factorization.