BIG DATA Summer 2020 - Assessment 2 Report - Group 14

Considering social, economic and political variables, can we predict whether a country is happy or not?

1.0 Introduction

The following report outlines the development and implementation of a model that predicts - in a binary format classification - the happiness of a country. There are a variety of surveys, and metrics used to define happiness. In this case a 'life ladder' variable is taken from the World Happiness Data issued by the United Nation's Sustainable Development Solutions Network (sourced from the Gallup world poll). The 'Life Ladder' value is derived from the Cantrail Ladder – obtained by asking participants to rate their current life using a scale on which ten is their best possible life and zero their worst. An average is then taken to give a score for 'Life Ladder'. If a country's score is above six, the population is deemed to be happy. This model could be useful in a range of situations; from diagnosing mental health conditions in a medical context, to making governmental policy changes and predicting the way in which a population will vote as a result.

2.0 Data Preparation

2.1 Data Information

The original dataset has 19 features and 1562 data entries. After visually analysing the data, it was discovered that NaN (not a number) values were spread amongst the data, rendering the data on those rows impractical. Two features had over 20% of its data as NaN values: gini of household income reported in Gallup, by wp5-year: 357 NaNs, and GINI index by World Bank estimate: 979 NaNs). They were both removed, and the rest of the sparse NaNs were eliminated by removing the rows on which they featured. After sanitisation of the data, our dataset had 17 features and 1120 data entries. A new feature was then created: 'LifeLadBin', with binary values, a score above six denoted happiness = 1, a score below six denoted unhappiness = 0 (Appendix 6.2).
Since people from 164 countries are surveyed over the years we plotted (Figure 1) the happiness trend from 2005 to 2017 for the countries we grouped into 10 regions.

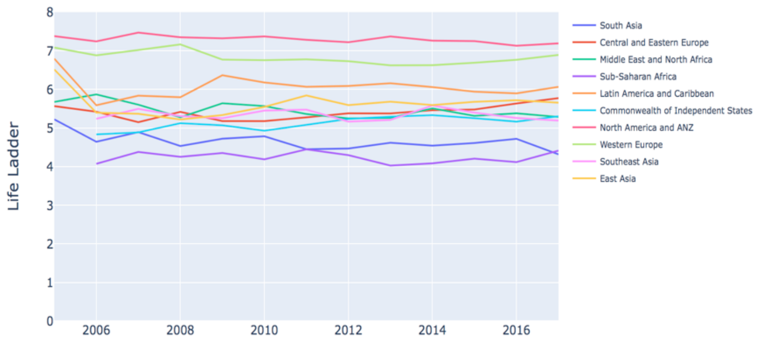


Figure 1 – Happiness trend fluctuation plot

The fluctuations were not very high and the happiness trends were mainly stable for each region. Therefore, we decided to place our focus on the countries without comparing it to the different years of observation.

2.2 Data Splitting

In order to build an accurate model that works on all data points without over-fitting, the original data set was split into three subsets; the training set contained 80% of the data, the validation and test sets contained 10% each. (Appendix 6.3).

2.3 Undersampling

A histogram was created (Figure 2) to display the distribution of the training set for descriptive analysis (Appendix 6.5 for the code).

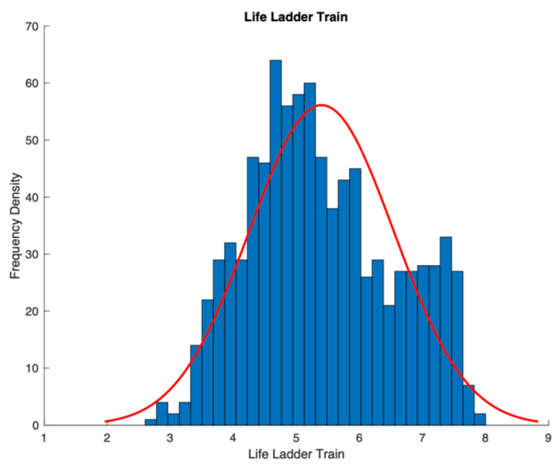


Figure 2– Descriptive analysis histogram

This showed that the training data for the 'Life Ladder' feature had a higher density at numbers lower than our threshold, and needed to be undersampled. 'Unhappy people' was the majority class, so this class was undersampled to balance the data. Before undersampling, there were 262 'happy' data points and 634 'unhappy' data points; after implementing the code (Appendix 6.4) there were 262 of both, meaning we had a balanced training set that would output a fair model. The same was done to the validation set.

3.0 Models

3.1 Logistic Regression

If a logistic regression analysis was carried out on the original unbalanced dataset, it could predict 'not happy' every time and achieve 70.8% accuracy. This is a benchmark to compare the final unbalanced test set to. However, as the training and validation set had been balanced, a higher accuracy needed to be achieved.

A backward selection model was also tested (Appendix 6.20). Variables were removed until there was no improvement to the accuracy of the model. However, when comparing the accuracies from the backward and forward selection models, the forward selection model gave better results and was used to select the features used in logistic regression (Appendix 6.7). The least absolute shrinkage and selection operator was also used with a L1-norm and changing the C value. The selection process (with (C=1e9)) was chosen to avoid having unnecessary features. This led to the selection of the 6 features listed below:

```
(['Healthy life expectancy at birth',
  'Positive affect',
  'Social support',
  'Generosity',
  'Standard deviation/Mean of ladder by country-year',
  'Standard deviation of ladder by country-year'],
```

The model built using these features produced 78.8% accuracy in the validation set (Appendix 6.8) and 83.0% accuracy in the test set (Appendix 6.9).

3.2 Decision Trees

Decision tree was used as a greedy model that made a locally optimal decision at each step, to get a global optima at the end. The impact of depth and the impurity on the accuracy of the model was plotted for both the training and validation sets (Appendix 6.10). As the model was made more complex, the accuracy of the training set kept on increasing while the validation accuracy was the highest between 3 and 5 features (Figure 3). Many combinations of depth were explored and a depth of 3 was chosen as it provided the best accuracy without overfitting the data.

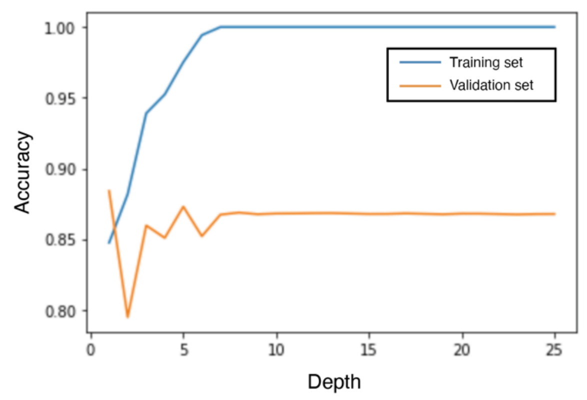


Figure 3 – Accuracy vs depth plot

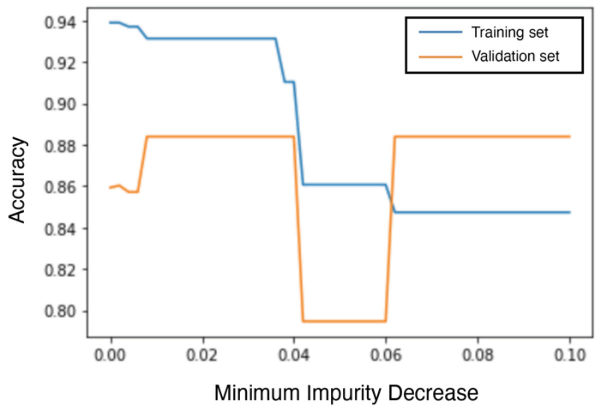


Figure 4 – Accuracy vs MID plot

The analysis started at the top of the tree and then going left or right according to the gini impurity that measures the probability of incorrect cases, this process gave us a "leaf node" with the lowest gini number (close to 0).

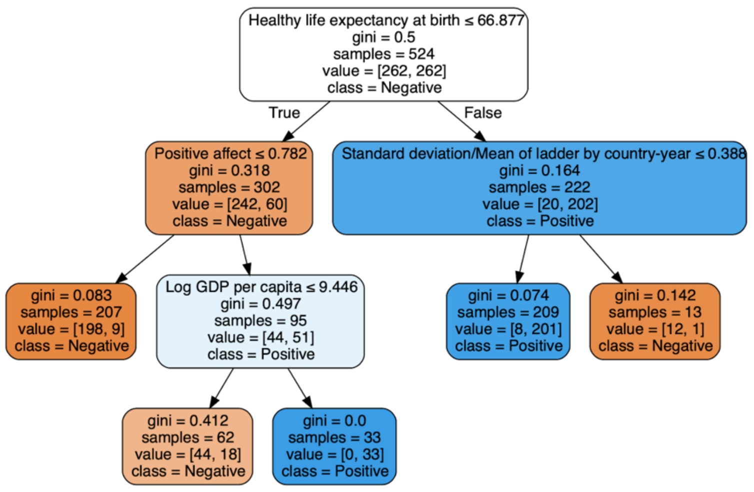


Figure 5 – Decision tree plot

3.3 Random Forests

To finally analyze the performance of our model, we ran the validation data sets with Random Forests. The Random Forests ran with a depth of 3 and with a minimum impurity decrease (MID) of 0.01 - many combinations of depth and MID were tested but the one which gave us the best accuracy in the end was kept. Our model achieved the following accuracies: 88.4% for the validation set and 87.5% for the test set (Appendix 6.16).

	Validation Set	Test Set
Precision	0.778	0.750
Accuracy	0.884	0.875
Recall	0.848	0.800

Table 1 – Results (more in Appendix 6.16)

3.4 Support Vector Machines

A support vector machine is a discriminative classifier, which allows us to validate the models we have used previously. Our validation data sets were run with SVM (using the hyper parameters C=1, gamma = 0.005 Appendix 6.17) and the following accuracies were achieved: 87.9% for the validation set (Appendix 6.18) and 78.6% for the test set (Appendix 6.19). These are similar values that our logistic regression model produced, validating our previous manual models.

4.0 Results and Analysis

Model	**Validation set accuracy**	**Test set accuracy**
Logistic Regression	78.8%	83.3%
Decision Tree	88.4%	58.9%
SVM	87.9%	78.6%
Random forest	88.4%	87.5%

Table 2 – Summary of findings

4.1 Logistic regression

Through forward selection of features, the logistic regression model showed that the following features were most influential in predicting happiness: ‘Healthy life expectancy at birth’, ‘Positive affect’, ‘Social support’, ‘Generosity’, ‘Standard deviation/Mean of ladder by country-year’ and ‘Standard deviation of ladder country-year’. The difference in accuracy between the validation and test set was only 4.2% - this shows that the model has not overfitted. The accuracy of this model on the test set was 83.0% - this is significantly higher than the original 70.8 %, but not the highest accuracy of all our models

4.2 Decision Tree

Based on gini values, the decision tree selected the following features: ‘Healthy life expectancy at birth’, ‘Positive affect’, ‘Standard deviation/Mean of ladder by country-year’ and ‘Log GDP per capita’. The first three features in this list are consistent with the logistic regression model. The training data was initially split between whether or not a country had a healthy life expectancy at birth. If that country did, the positive affect for that country was analysed, and a second split carried out. This process was continued for a depth of 3. The final ginis were in a range of 0 to 0.412 - there were two pure leaf nodes (ginis of 0.0 and 0.074) whereas the rest were impure. The accuracy on the validation set for the tree was high (88.4%), whereas the accuracy for the test set was relatively low (58.9%). This implies that the tree was potentially over-fitting, which is addressed in the implementation of Random Forests.

4.3 Random forests

The random forest model was created to assess the quality of the selection of features of the decision tree shown above. The random forest model runs a number of randomly created decision trees (the optimal number of tree number is 14 and was computed in the Appendix 6.14) and combines the results, to determine the most accurate prediction. Our random forest model gave an accuracy of 87.5% for the test set, which was higher than that of the decision tree. This is predictable due to the tendency of decision trees to over fit.

4.4 Analysis

Our models consistently show that having a healthy life expectancy at birth is the most significant feature (validated with a relatively low p-value: Appendix 6.25) when predicting whether or not a population will be happy. However, it is also interesting to consider the other features highlighted by forward selection, such as social support and generosity - the prominence of these features imply that physical and mental support in society play a larger role in the happiness of a population than other more concrete metrics, such as ‘Log GDP per capita’. This information could prove useful for a government looking to make changes to improve the general happiness of its people.

The limitations of our model include the fact that it is only applicable to the countries and areas involved in the data collection - whilst the list is extensive, it is not quite worldwide (there are 195 countries in the world as opposed to the 164 in the dataset). The use of 19 specific features is also a limitation, as the results of our models can only predict happiness based on these metrics, so a wider set of features (like access to education, medical services...) in the initial survey would have maybe improved our model.

Finally, using a category system other than the binary ‘happy’ or ‘not happy’ would have been more insightful, with our model predicting happiness on a scale, more conclusions could be drawn. For example, making the happiest country: Finland, a benchmark to compare it to other happy countries and seeing what social, economic and political features made the overall difference.

However, we were constrained by our research that provided us with the information that a Life Ladder score of 6 is the threshold above which people are deemed happy - for this reason we used a binary system, rather than a continuous.

5.0 Conclusion

After many trials, we have designed a model which gives improved predictions of whether a population is happy or not, and the factors that affect happiness most were uncovered. To get a tangible visualization of the life ladder distribution as well as its correlation with the socio-economic and geopolitical context, we have created a geographical visualization of happiness with a heat map with a gradient range between 3 and 8 (Figure 6).

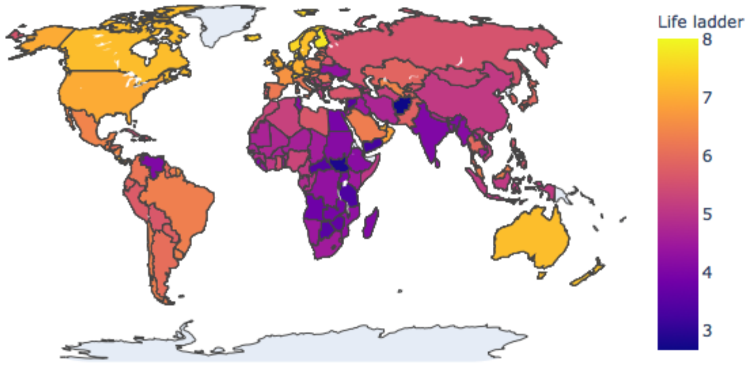


Figure 6 – Happiness heat map

Another heat map was plotted with the binary life ladder showing the countries that are happy in yellow and unhappy in blue - Appendix 6.21.

We have established a model that produces more false negatives and a lower recall. This result is better than getting more false positives because it is safer to predict that people are unhappy to push governments and the United Nations to act by reducing conflict and achieving peace for a better well-being of the population.

Another interesting application of this model could be uncovered as a result of recent events - in the wake of Covid-19, the dire economic situation that many countries face could lead to a severe decrease in happiness globally due to factors such as job loss and inability to travel. Our model could help governments to consider and include other factors to improve general happiness, and therefore maintain the population’s wellbeing.

Word count: 1954

6.0 Appendix

6.1 Data source

1. Yeung A. World Happiness Data [Internet]. Kaggle.com. 2020 [cited 19 June 2020]. Available from: https://www.kaggle.com/arielyeung/world-happiness-data?fbclid=IwAR1xksU8-REUj_FEji1eWBVFoVIpXvf5lyPw5YN5j-iE3_Zfij89LvTPXYE

6.2 Data preparation code

```
#read the csv file
happiness = pd.read_csv('Original_2017_full.csv')

#Inspect which columns have a significant number of NaNs
for i in range(0,19):
    print('NaNs in {}: {}'.format(happiness.columns[i], happiness[happiness.columns[i]].isna().sum()))

#Drop columns which consist of over 20% NaNs
happiness = happiness.drop(columns=['gini of household income reported in Gallup, by wp5-year','GINI index (World Bank estimate)'])

#Drop data entries which have NaNs
happiness.dropna(inplace = True)

#Showing that rows with NaNs had been dropped for all essential columns
for i in range(0,17):
    print('NaNs in {}: {}'.format(happiness.columns[i], happiness[happiness.columns[i]].isna().sum()))

#Creating a new column called 'LifeLadBin'; 'LifeLadBin' = 1 if the 'Life Ladder' column is above 6
happiness['LifeLadBin'] = happiness['Life Ladder'] > 6
happiness['LifeLadBin'] = happiness['LifeLadBin'].astype(int)
```

6.3 Splitting the data

```
#Splitting the main dataset into train validation and test sets
train, other = train_test_split(happiness, test_size=0.2, random_state=0)
validation, test = train_test_split(other, test_size=0.5, random_state=0)
```

6.4 Undersampling the data

```
#Making the number of positive and negative samples in the train and val data sets equal to correct the imbalancing
happyt =  (train['LifeLadBin']==1).sum()
totalt = len(train)
nothappyt = totalt - happyt
happyv =  (validation['LifeLadBin']==1).sum()
totalv = len(validation)
nothappyv = totalv - happyv

#Too many negative samples, so number of negative samples cut to match positive samples
happy_post = train.loc[train['LifeLadBin'] == 1]
happy_negt = train.loc[train['LifeLadBin'] == 0].sample(happyt)
happy_posv = validation.loc[validation['LifeLadBin'] == 1]
happy_negv = validation.loc[validation['LifeLadBin'] == 0].sample(happyv)

#Create train and validation datasets with equal number of positive and negative samples
resampledtrain = pd.concat((happy_negt, happy_post))
resampledval = pd.concat((happy_negv, happy_posv))

happynewt =  (resampledtrain['LifeLadBin']==1).sum()
totalnewt = len(resampledtrain)
nothappynewt = totalnewt - happynewt
happynewv =  (resampledval['LifeLadBin']==1).sum()
totalnewv = len(resampledval)
nothappynewv = totalnewv - happynewv
```

6.5 Histogram Matlab code

```
m = csvread('lifeladdertrain.csv')
hold on
title('Life Ladder Train')
xlabel('Life Ladder')
xlabel('Life Ladder Train')
ylabel('Frequency Density')
histfit(m(:,1))
hold off
```

6.6 Creating ‘x’ and ‘y’ values for each data set

```
X_train = resampledtrain.drop(columns=['LifeLadBin','Life Ladder','country'])
y_train = resampledtrain['LifeLadBin']

X_val = resampledval.drop(columns=['LifeLadBin','Life Ladder','country'])
y_val = resampledval['LifeLadBin']

X_test = test.drop(columns=['LifeLadBin','Life Ladder','country'])
y_test = test['LifeLadBin']
```

6.7 Forward selection

```
columns_in_model = ['Healthy life expectancy at birth','Positive affect','Social support']
columns_to_test = X_train.columns

def select_column_to_add(X_train, y_train, X_val, y_val, columns_in_model, columns_to_test):

    column_best = None
    columns_in_model = list(columns_in_model)

    if len(columns_in_model) == 0:
        acc_best = 0
    elif len(columns_in_model) == 1:
        mod = LogisticRegression(C=1e9).fit(X_train[columns_in_model].values.reshape(-1, 1), y_train)
        acc_best = accuracy_score(y_val, mod.predict(X_val[columns_in_model].values.reshape(-1, 1)))
    else:
        mod = LogisticRegression(C=1e9).fit(X_train[columns_in_model], y_train)
        acc_best = accuracy_score(y_val, mod.predict(X_val[columns_in_model]))


    for column in columns_to_test:
        mod = LogisticRegression(C=1e9).fit(X_train[columns_in_model+[column]], y_train)
        y_pred = mod.predict(X_val[columns_in_model+[column]])
        acc = accuracy_score(y_val, y_pred)

        if acc - acc_best >= 0.005:  # one of our stopping criteria
            acc_best = acc
            column_best = column

    if column_best is not None:  # the other stopping criteria
        print('Adding {} to the model'.format(column_best))
        print('The new best validation accuracy is {}'.format(acc_best))
        columns_in_model_updated = columns_in_model + [column_best]
    else:
        print('Did not add anything to the model')
        columns_in_model_updated = columns_in_model

    return columns_in_model_updated, acc_best

select_column_to_add(X_train, y_train, X_val, y_val, columns_in_model, columns_to_test)
```

6.8 Logistic regression model results on validation set

```
Accuracy: 0.788
Precision: 0.757
Recall: 0.848
----------------------------------------------------------
Confusion Matrix (total:66)      Accuracy:        0.788
  TP: 28 | FN: 5
  FP: 9 | TN: 24
```

6.9 Logistic regression model results on test set

```
Accuracy: 0.830
Precision: 0.923
Recall: 0.400
----------------------------------------------------------
Confusion Matrix (total:112)      Accuracy:        0.83
  TP: 12 | FN: 18
  FP: 1 | TN: 81
```

6.10 Testing the maximum depth and the minimum impurity decrease

6.10.1 Depth graph code

```python
import matplotlib.pyplot as plt

lista = list(range(1,26))
listtrain = []
listval = []
count = list(range(1,1001))
number = 0

for x in lista:
    temptrain = []
    temp = []
    for y in count:
        model = tree.DecisionTreeClassifier(max_depth=x, min_impurity_decrease = 0)
        model.fit(X_train,y_train)
        y_pred = model.predict(X_val)
        y_pred_train = model.predict(X_train)
        acc_train = accuracy_score(y_train,y_pred_train)
        acc = accuracy_score(y_val,y_pred)
        prec = precision_score(y_val,y_pred)
        rec = recall_score(y_val,y_pred)
        temptrain.append(acc_train)
        temp.append(acc)
        if y == 1000:
            meantrain = sum(temptrain)/len(temptrain)
            meanval = sum(temp)/len(temp)
            listtrain.append(meantrain)
            listval.append(meanval)
            number = number + 1
            print (number)

plt.plot(lista,listtrain)
plt.plot(lista,listval)
```

6.10.2 MID graph code

```python
plt.plot(lista,listtrain)
plt.plot(lista,listval)

lista = np.linspace(0,0.1,51)
listtrain = []
listval = []
count = list(range(1,101))
number = 0

for x in lista:
    temptrain = []
    temp = []
    for y in count:
        model = tree.DecisionTreeClassifier(max_depth=3, min_impurity_decrease = x)
        model.fit(X_train,y_train)
        y_pred = model.predict(X_val)
        y_pred_train = model.predict(X_train)
        acc_train = accuracy_score(y_train,y_pred_train)
        acc = accuracy_score(y_val,y_pred)
        prec = precision_score(y_val,y_pred)
        rec = recall_score(y_val,y_pred)
        temptrain.append(acc_train)
        temp.append(acc)
        if y == 100:
            meantrain = sum(temptrain)/len(temptrain)
            meanval = sum(temp)/len(temp)
            listtrain.append(meantrain)
            listval.append(meanval)
            number = number + 1
            print (number)

plt.plot(lista,listtrain)
plt.plot(lista,listval)
```

6.11 Decision tree iterations

```python
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier

clf1 = tree.DecisionTreeClassifier(max_depth=4, min_impurity_decrease=0.001)
clf1.fit(X_train, y_train)
print('Accuracy on val set: {}'.format(accuracy_score(y_test, clf1.predict(X_val))))

clf2 = tree.DecisionTreeClassifier(max_depth=3, min_impurity_decrease=0.001)
clf2.fit(X_train, y_train)
print('Accuracy on val set: {}'.format(accuracy_score(y_test, clf2.predict(X_val))))

clf3 = tree.DecisionTreeClassifier(max_depth=3, min_impurity_decrease=0.01)
clf3.fit(X_train, y_train)
print('Accuracy on test set: {}'.format(accuracy_score(y_test, clf3.predict(X_val))))
```

6.12 Decision tree visualization

```python
import sklearn.tree as tree
import graphviz

dot_data = tree.export_graphviz(clf3, out_file=None)
graph = graphviz.Source(dot_data)

predictors = X_train.columns
dot_data = tree.export_graphviz(clf3, out_file=None,
                                feature_names = predictors,
                                class_names = ('Negative', 'Positive'),
                                filled = True, rounded = True,
                                special_characters = True)
graph = graphviz.Source(dot_data)
```

6.13 Random Forest Model code

```python
#random forest with a max depth of 3 to assess the accuracy of the decision tree
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(max_depth = 3,min_samples_split = 10)
model = model.fit(X_train,y_train)

y_pred_val = model.predict(X_val)
acc_val = accuracy_score(y_val,y_pred_val)
prec_val = precision_score(y_val,y_pred_val)
rec_val = recall_score(y_val,y_pred_val)

y_pred = model.predict(X_test)
acc_test = accuracy_score(y_test,y_pred)
prec_test = precision_score(y_test,y_pred)
rec_test = recall_score(y_test,y_pred)

print ('Validation set')
print ('Precision: {}'.format(prec_val))
print ('Accuracy: {}'.format(acc_val))
print ('Recall: {}'.format(rec_val))
confusion_matrix_val = confusion_matrix(y_val,y_pred)
print (confusion_matrix_val)
print ('Test set')
print ('Precision: {}'.format(prec_test))
print ('Accuracy: {}'.format(acc_test))
print ('Recall: {}'.format(rec_test))
confusion_matrix_test = confusion_matrix(y_test,y_pred)
print (confusion_matrix_test)
```

6.14 Determination of optimal number of trees in forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

%matplotlib qt

best_model = None
max_validation_accuracy = 0

AccuracyTraining = []
AccuracyValidation = []
ka = []

for k in range (1):
    model = RandomForestClassifier(random_state=2, n_estimators=14, max_depth=9, min_impurity_decrease = 0.0001)
    ka.append(k+1)

    model.fit(X_train,y_train)

    y_pred = model.predict(X_train)
    accuracy = accuracy_score(y_train, y_pred)
    AccuracyTraining.append(accuracy)

    y_pred = model.predict(X_val)
    accuracy = accuracy_score(y_val, y_pred)
    AccuracyValidation.append(accuracy)

    if accuracy > max_validation_accuracy:
        max_validation_accuracy = accuracy
        best_model = model

print('Optimal tree number: {}'.format(len(best_model.estimators_)))

plt.plot(ka,AccuracyTraining)
plt.plot(ka,AccuracyValidation)

Forest = best_model
```

6.15 Accuracy for the decision tree

```
Accuracy on val set: 0.8839285714285714
```

6.16 Random forest results

```
Validation set
Precision: 0.7777777777777778
Accuracy: 0.8839285714285714
Recall: 0.8484848484848485
[[56 23]
 [24  9]]
Test set
Precision: 0.75
Accuracy: 0.875
Recall: 0.8
[[74  8]
 [ 6 24]]
```

6.17 Support Vector Machine code

```
from sklearn.svm import SVC

model = SVC(C=1, kernel='rbf', gamma = 0.005)
model = model.fit(X_train, y_train)
ypred = model.predict(X_val)

acc = accuracy_score(y_val,ypred)
prec = precision_score(y_val,ypred)
rec = recall_score(y_val,ypred)
print('Precision: {}'.format(prec))
print('Recall: {}'.format(rec))
print('Accuracy:{}'.format(acc))
```

6.18 SVM results for validation set

```
Precision: 0.9310344827586207
Recall: 0.8181818181818182
Accuracy:0.8787878787878788
```

6.19 SVM results for test set

```
Precision: 0.5681818181818182
Recall: 0.8333333333333334
Accuracy:0.7857142857142857
```

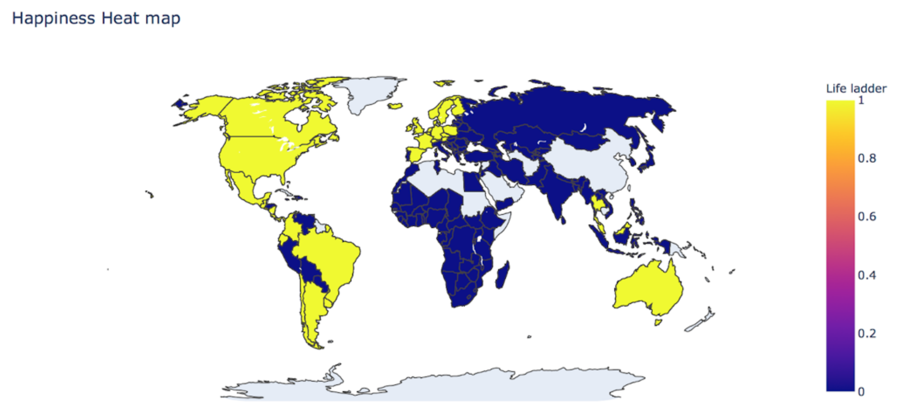
6.20 Backward selection

```
y = resampledtrain['LifeLadBin'].values.reshape(-1,1)
#x = resampledtrain[['Log GDP per capita', 'Positive affect', 'Perceptions of corruption']]

x = resampledtrain[['Log GDP per capita', 'Social support', 'Healthy life expectancy at birth', 'Freedom to mal
```

```
y = resampledtrain['LifeLadBin'].values.reshape(-1,1)
#x = resampledtrain[['Log GDP per capita', 'Positive affect', 'Perceptions of corruption']]

x = resampledtrain[['Log GDP per capita', 'Social support', 'Healthy life expectancy at birth', 'Freedom to mal
```

6.21 Heat Map

```
from plotly.offline import init_notebook_mode, iplot
import plotly.graph_objs as go
data = dict(type = 'choropleth',
           locations = happiness['country'],
           locationmode = 'country names',
           z = happiness['LifeLadBin'],
           colorbar = {'title':'Life ladder'})
layout = dict(title = 'Happiness Heat map',
             geo = dict(showframe = False,
                        projection = {'type': 'natural earth'}))
choromap = go.Figure(data = [data], layout = layout)
iplot(choromap)
```



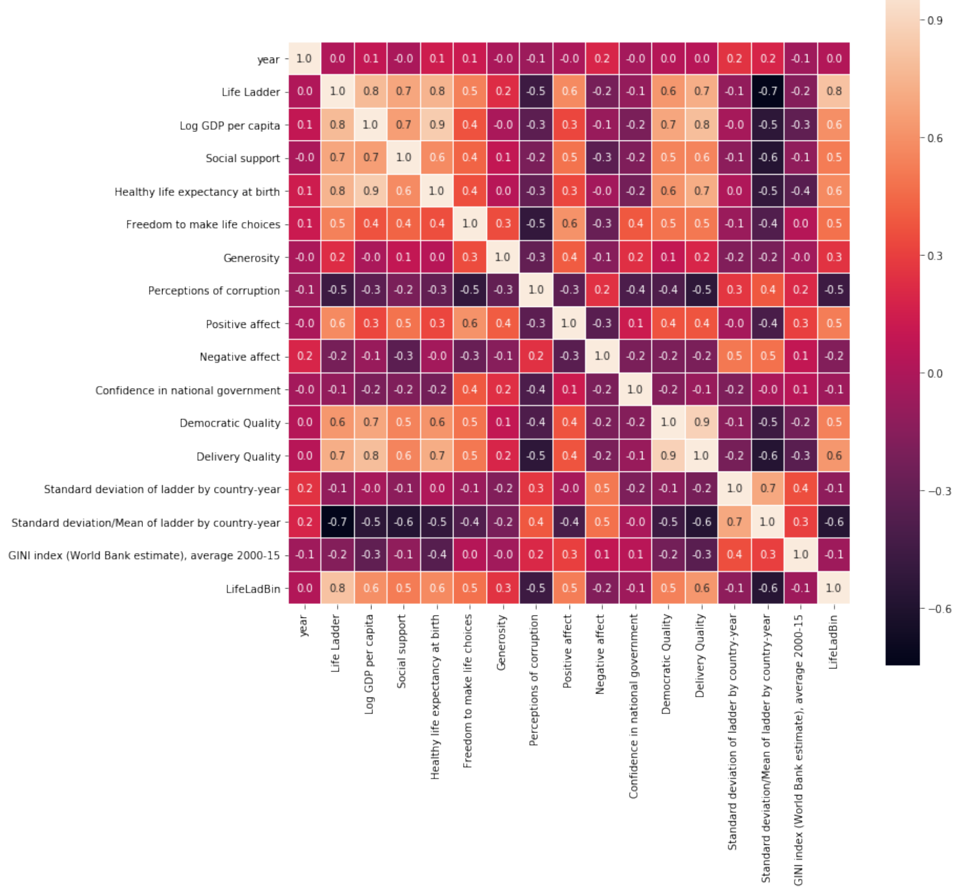
6.22 Table of mentioned features and their meaning

Feature	Meaning
Life Ladder (0-10)	Derived from the Cantrail ladder - to obtain values for the Cantrail ladder participants are asked to imagine a scale on which ten is their best possible life and zero is their worst possible life.
Healthy life expectancy at birth (0-100)	The average equivalent number of years of full health that a newborn could expect to live, if he or she were to pass through life subject to the age-specific death rates and ill-health rates of a given period (based on data from the World Health Organization (WHO) Global Health Observatory data repository).
Positive affect (0-1)	Defined as the average frequency of happiness, laughter and enjoyment on the previous day.
Negative affect (0-1)	Defined as the average frequency of worry, sadness and anger on the previous day.
Social support (0-1)	The national average of the binary responses (either 0 or 1) to the Gallup World Poll (GWP) question “If you were in trouble, do you have relatives or friends you can count on to help you whenever you need them, or not?”
Generosity (-1;1)	The residual of regressing the national average of GWP responses to the question “Have you donated money to a charity in the past month?” on GDP per capita.
Standard deviation/Mean of ladder by country-year	The standard deviation of the Life Ladder feature, divided by the mean of that feature. (No generic scale)
Standard deviation of ladder by country-year	The standard deviation of the Life Ladder feature. (No generic scale)
Log GDP per capita	The natural log of GDP per capita - the average income earned per person in a given area in a specific year.

6.23 Features heatmap matrix

2D visualization to show the correlation between the different features and get a first insight onto which ones affect the happiness index.

```
import seaborn as sns
f,ax = plt.subplots(figsize = (12, 12))
sns.heatmap(happiness.corr(), annot = True, linewidths = 0.1, fmt = '.1f', ax = ax, square = True)
```



6.24 Code to plot Figure 1 - Happiness trend fluctuation

```
layout = go.Layout(title="Happiness Trend over the years", font=dict(size=18),
                   xaxis=dict(title='Year', titlefont=dict(size=18),
                              tickfont=dict(size=14)),
                   yaxis=dict(range=[0, 8], title='Life Ladder',
                              titlefont=dict(size=18), tickfont=dict(size=14)),
                   legend=dict(font=dict(size=10)))
fig = {'data': [{'x': df_allyear[df_allyear['region'] == region].groupby('year')
                 .agg({'happiness': 'mean'}).reset_index()['year'],
                 'y': df_allyear[df_allyear['region'] == region].groupby('year')
                 .agg({'happiness': 'mean'}).reset_index()['happiness'],
                 'name': region, 'mode': 'lines', } for region in df_allyear['region'].unique()],
       'layout': layout}
py.iplot(fig)
```

6.25 P-value and descriptive values

```
                                                    coefficient    std  \
intercept                                                -1.986  4.097
Healthy life expectancy at birth                          0.050  0.040
Positive affect                                           0.744  2.885
Social support                                           -0.372  3.323
Generosity                                                1.044  1.514
Standard deviation/Mean of ladder by country-year        -1.949  4.994
Standard deviation of ladder by country-year             -0.797  1.223

                                                    p-value  [0.025  0.975
]
intercept                                             0.628 -10.105   6.13
2
Healthy life expectancy at birth                      0.208  -0.029   0.12
9
Positive affect                                       0.796  -4.972   6.46
1
Social support                                        0.911  -6.957   6.21
3
Generosity                                            0.490  -1.955   4.04
4
Standard deviation/Mean of ladder by country-year     0.696 -11.846   7.94
8
Standard deviation of ladder by country-year          0.514  -3.220   1.62
6
```