Data Mining

Health Analytics

Background and introduction

One of the common application areas of data mining is in health. Medical organisations often store extensive data on patient medical history, conditions, personal characteristics such as sex, age, height etc. and lifestyle choices such as smoking, drinking and exercise. In most cases, this data is not effectively utilised. Using data mining, medical organisations can take advantage of this data by building models to extract useful patterns and information from it.

For example, these models can be used to estimate or predict the health of an individual patient, given the data that is known about them. Patients at high risk can be identified, and preventative action can be taken. In addition, when treating patients, data mining models have the potential to identify courses of treatment that have been effective for similar patients.

In this report, a set of NHS patient data is considered. The data contains features such as age, sex, height, weight, units of alcohol drunk, cigarettes smoked, levels of exercise and overall health score for 5000 patients. The data is first analysed in order to answer questions such as:

- "Are there significant differences between different segments of the population in terms of their lifestyle choices?"
- "Which individual characteristics and lifestyle choices impact a person's health score, and to what extent?"
- "What would be the impact on the health scores of the overall population if nobody consumed alcohol or smoked?"

Based on this analysis, a set of data mining models have been developed and applied to the data, to predict the health scores of 20 unseen individuals, based on the other features in the dataset. The results are then discussed. The models developed are a random forest model, neural network, linear regression model and support vector machine.

Data analytics

(Note: the Python notebook used for performing the data analytics can be found within the ZIP file submitted.)

For the analysis of the data, the first step has been to read in the health score data (5000 individuals) and population data (20 individuals) into Pandas data frames. Pandas data frames are commonly used for handling data in Python, because they have a number of useful and convenient features for dealing with data (e.g. index by column to get all of a given feature from the data, automatically create correlation matrices, calculate mean, create histograms etc).

In [2]:
```
#Import the health score data into a pandas DataFrame and display the first 5 and last 5 values
healthScores = pd.read_csv("HealthScores.csv")
healthScores.head()
```

Out[2]:

	Age	Sex	Weight in lbs	Height in Inch	IQ	Units of alcohol per day	Cigarettes per day	Active	Health Score (high is good)
0	32	Female	101	67	110	0	0	Inactive	251
1	47	Female	138	67	84	2	0	Active	193
2	25	Male	166	73	106	2	5	Inactive	167
3	75	Male	184	73	99	2	0	Inactive	93
4	72	Female	105	65	99	2	0	Active	150

In [3]:
```
healthScores.tail()
```

Out[3]:

	Age	Sex	Weight in lbs	Height in Inch	IQ	Units of alcohol per day	Cigarettes per day	Active	Health Score (high is good)
4994	56	Female	77	64	98	3	0	Inactive	133
4995	38	Male	151	70	103	5	0	Inactive	100
4996	45	Female	135	62	99	0	0	Very Active	258
4997	72	Female	130	62	102	0	0	Inactive	125
4998	26	Female	142	68	99	0	0	Very Active	378

Next, the calculated value BMI was added to the data. Weight was converted to kg, height was converted to metres, and the BMIs were calculated using the formula:

$$BMI = \frac{Weight}{Height^2}$$

BMI is a measure intended to indicate general health, so it was added because it may be a useful predictor of health scores, and can be calculated from the data already available.

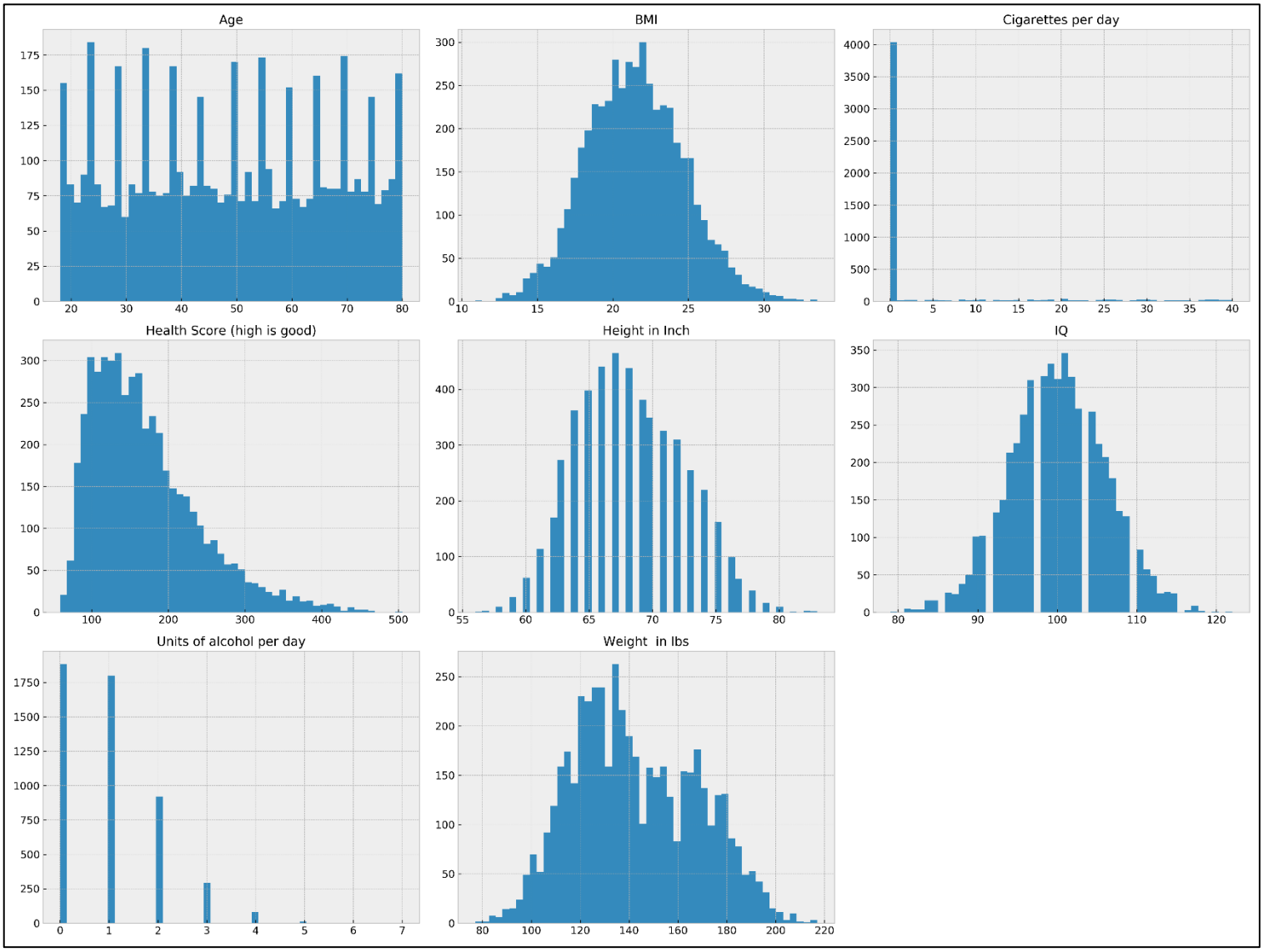
In [28]:
```
#Add a column (feature) to the health score data set for BMI (BMI is a value calculated from weight and height)

#BMI = weight (in kg) / (height (in m))^2
#See https://www.nhs.uk/chq/Pages/how-can-i-work-out-my-bmi.aspx

#copy healthScores data frame into a temporary data frame
temp = healthScores.copy()

#convert weights from lbs to kg
temp['Weight in kg'] = temp['Weight  in lbs'] * 0.453592

#convert heights from inches to metres
temp['Height in m'] = temp['Height in Inch'] * 0.0254

#calculate BMIs and store them in a new column titled "BMI"
healthScores['BMI'] = temp['Weight in kg'] / (temp['Height in m'] **2)

healthScores.head()
```

Out[28]:

	Age	Sex	Weight in lbs	Height in Inch	IQ	Units of alcohol per day	Cigarettes per day	Active	Health Score (high is good)	BMI
0	32	Female	101	67	110	0	0	Inactive	251	15.818661
1	47	Female	138	67	84	2	0	Active	193	21.613616
2	25	Male	166	73	106	2	5	Inactive	167	21.900817
3	75	Male	184	73	99	2	0	Inactive	93	24.275605
4	72	Female	105	65	99	2	0	Active	150	17.472721

A number of histograms have been plotted of the different features in the data set. There is a flat distribution on age, although certain ages seem to be more represented in the data set than others. BMI, IQ, height and weight are all normally distributed, as might be expected. The vast majority of people do not smoke, and the histogram reflects this. Both alcohol and health score have a positively skewed normal distribution. The majority of people drink few units of alcohol, and ever-decreasing numbers of people drink more than one unit per day. For health scores, the majority of people have a health score below 200, with ever-decreasing numbers of people having health scores higher than this.

```
In [32]: %matplotlib inline
         import matplotlib.pyplot as plt
         healthScores.hist(bins=50, figsize=(20,15))
         save_fig("attribute_histogram_plots")
         plt.show()

         Saving figure attribute_histogram_plots
```

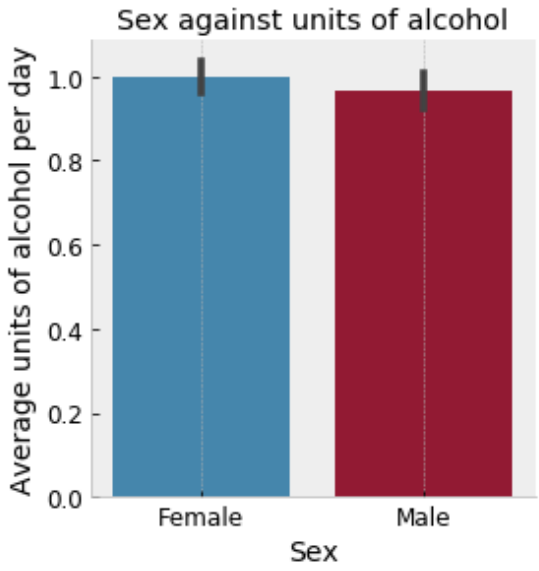


Are there significant differences between different segments of the population in terms of their lifestyle choices?

In order to answer this question, only different genders and age groups will be considered as different segments of the population. Different heights, weights and IQs will not be considered different segments of the population. A number of graphs have been plotted. Sex has been plotted against units of alcohol per day, cigarettes per day and daily activity level. Age has also been plotted against units of alcohol, cigarettes and daily activity level.

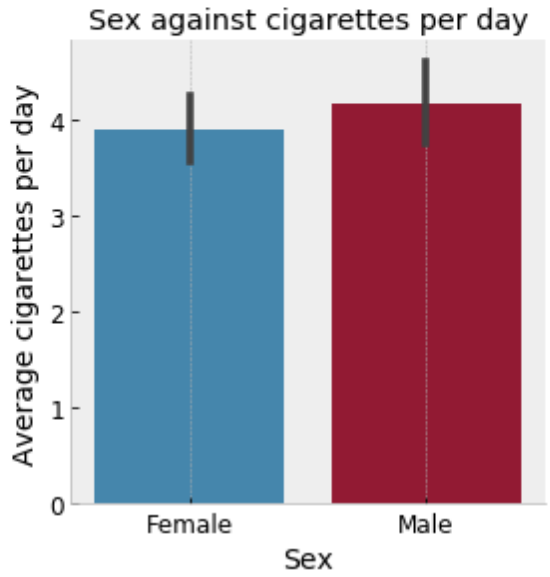
In the plot below of sex against units of alcohol, it can be seen that women drink very slightly more than men, averaging one unit per day, as opposed to around 0.98 units for men. There is no significant difference here.

```
[34]: #Sex against units of alcohol drunk
sns.factorplot("Sex", "Units of alcohol per day", data=healthScores, kind="bar")

plt.title('Sex against units of alcohol')
plt.ylabel('Average units of alcohol per day')
plt.grid()
```



In the plot below of sex against cigarettes smoked, it can be seen that men smoke slightly more than women, averaging about 4.2 cigarettes per day, compared to about 3.9 for women. It should be noted that the vast majority of people do not smoke, so the average of about 4 per day is mostly due to a limited number of individuals who smoke extremely high numbers of cigarettes per day.

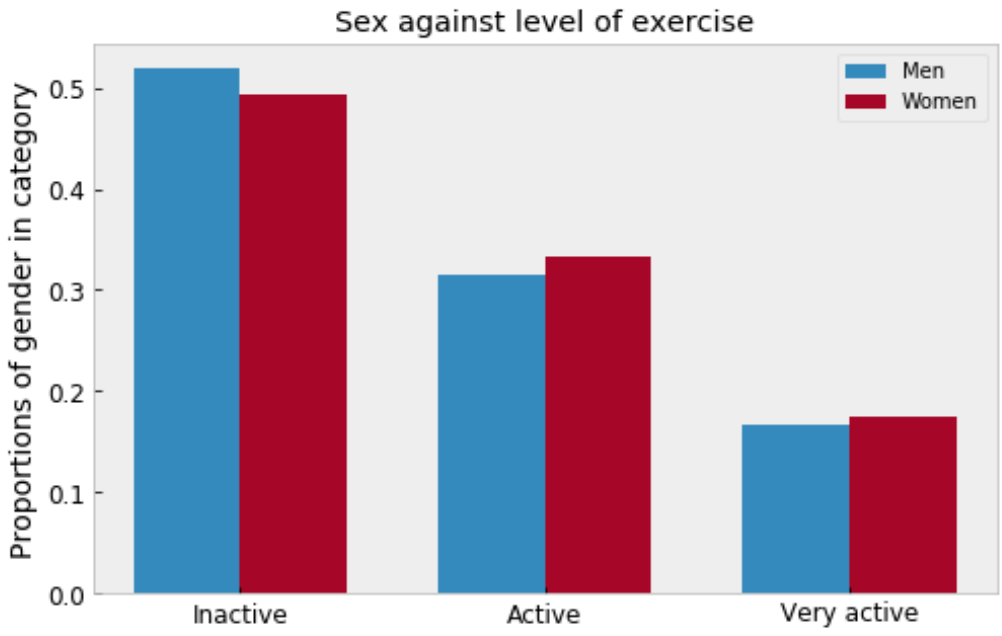
In [35]:
```
#Sex against cigarettes per day
sns.factorplot("Sex", "Cigarettes per day", data=healthScores, kind="bar")

plt.title('Sex against cigarettes per day')
plt.ylabel('Average cigarettes per day')
plt.grid()
```



The code below splits the data into two separate data frames, one for the males and one for the females. It then calculates the proportion of men who fall into each activity level category of “inactive, “active” and “very active” (the proportions could be 0.5, 0.3, 0.2, for example). It does the same for the women. The bars for each sex are then plotted.

In the chart, it can be seen that for both males and females, around 50% are inactive, around 30% are active and around 20% are very active. A slightly higher proportion of males are inactive than females. Slightly more females are active or very active than males. The differences are not particularly significant, however.

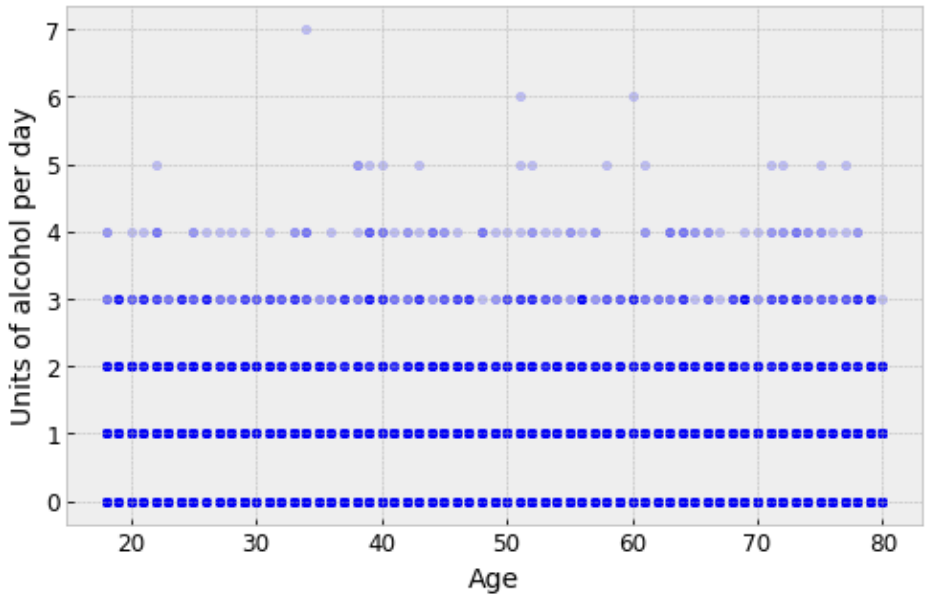
In [52]:
```
#Sex against activity level

males = healthScores.loc[healthScores['Sex'] == 'Male']
females = healthScores.loc[healthScores['Sex'] == 'Female']

#Proportions of men and women who fall into each activity level
maleCounts = males['Active'].value_counts(normalize=True)
femaleCounts = females['Active'].value_counts(normalize=True)

fig, subplot = plt.subplots(figsize=(8,5))

ind = np.arange(3)
p1 = subplot.bar(ind, maleCounts, 0.35, bottom=0)
p2 = subplot.bar(ind + 0.35, femaleCounts, 0.35, bottom=0)

subplot.set_title('Sex against level of exercise')
subplot.grid()
subplot.set_xticks(ind + 0.35 / 2)
subplot.set_xticklabels(('Inactive', 'Active', 'Very active'))
subplot.set_ylabel('Proportions of gender in category')

subplot.legend((p1[0], p2[0]), ('Men', 'Women'))
subplot.autoscale_view()
```



The chart below is a scatter plot of age against units of alcohol per day. The data available rounds the units of alcohol drunk per day to integer values, which is the reason for the discrete rows in the scatter plot. It can be seen that regardless of age, most people drink 0, 1 or 2 units per day. Among those who drink 3 or more units, there is no apparent correlation with age either.

```
: #Age against units of alcohol drunk

  healthScores.plot(kind='scatter', x='Age', y='Units of alcohol per day', alpha=0.2, figsize=(8, 5))
```

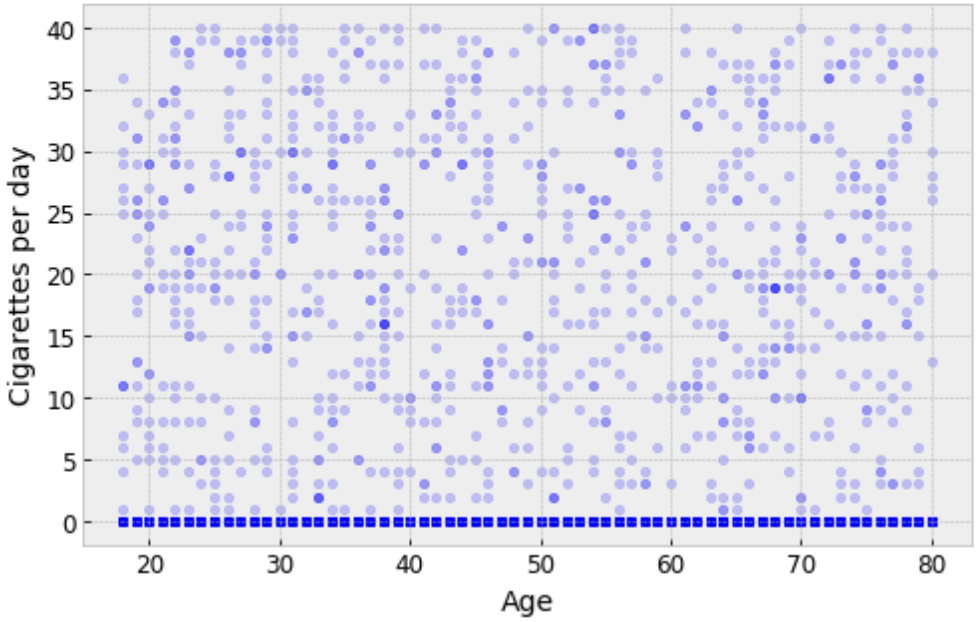
```
: <matplotlib.axes._subplots.AxesSubplot at 0x1985e1b76d8>
```



In the scatter plot below, age is plotted against number of cigarettes per day. No correlation is found between the two variables. Regardless of age, the majority of people do not smoke. Those who do smoke are broadly distributed with regard to age.

```
#Age against cigarettes per day

healthScores.plot(kind='scatter', x='Age', y='Cigarettes per day', alpha=0.2, figsize=(8, 5))
```

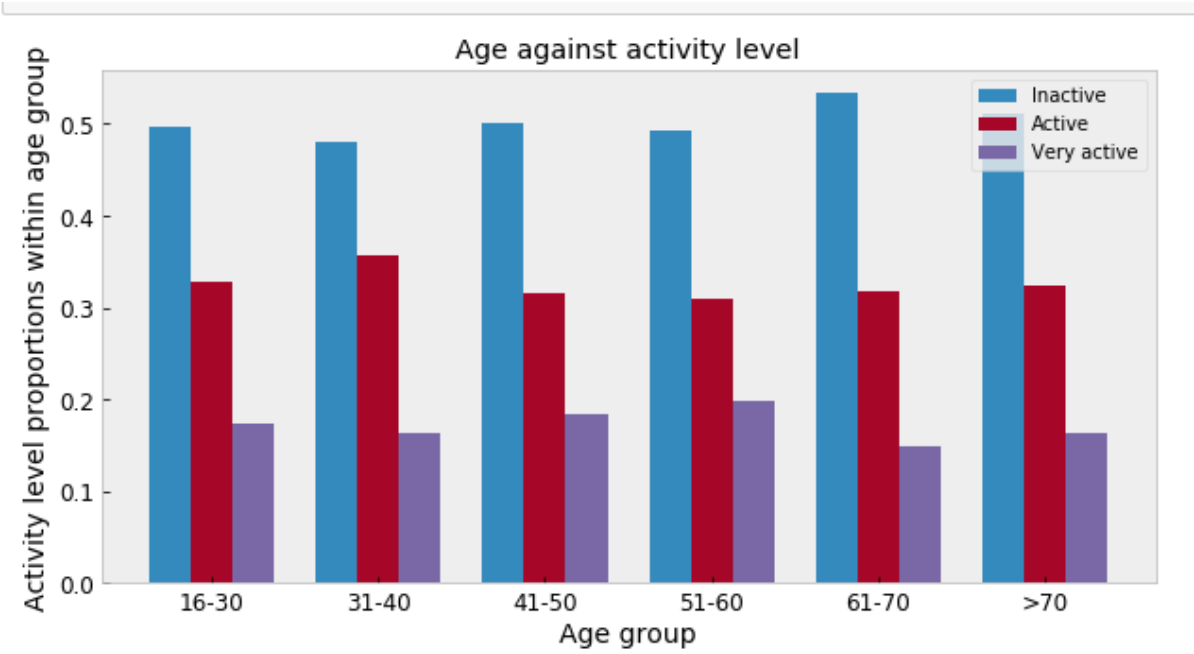
```
<matplotlib.axes._subplots.AxesSubplot at 0x1985bf89278>
```



The code below is used to plot a grouped bar chart that compares age against exercise level. The code works out the exercise level proportions within each age group. First, the data is split into age groups, and then it is proportionalised and plotted. Each different column colour represents an activity level

```
#Age against activity level
#Age groups are 16-30, 31-40, 41-50, 51-60, 61-70, 71-80

groups = []
groups.append(healthScores.loc[(healthScores['Age'] >= 16) & (healthScores['Age'] <= 30)])
groups.append(healthScores.loc[(healthScores['Age'] > 30) & (healthScores['Age'] <= 40)])
groups.append(healthScores.loc[(healthScores['Age'] > 40) & (healthScores['Age'] <= 50)])
groups.append(healthScores.loc[(healthScores['Age'] > 50) & (healthScores['Age'] <= 60)])
groups.append(healthScores.loc[(healthScores['Age'] > 60) & (healthScores['Age'] <= 70)])
groups.append(healthScores.loc[healthScores['Age'] > 70])

counts = []
for i in range(0, 6):
    counts.append(groups[i]['Active'].value_counts(normalize=True))

inactiveSubplot = []
for i in range(0, 6):
    inactiveSubplot.append(counts[i]['Inactive'])

activeSubplot = []
for i in range(0, 6):
    activeSubplot.append(counts[i]['Active'])

veryActiveSubplot = []
for i in range(0, 6):
    veryActiveSubplot.append(counts[i]['Very Active'])

fig, subplot = plt.subplots(figsize=(10, 5))

ind = np.arange(6)
inactive = subplot.bar(ind, inactiveSubplot, 0.25, bottom=0)
active = subplot.bar(ind + 0.25, activeSubplot, 0.25, bottom=0)
veryActive = subplot.bar(ind + 0.5, veryActiveSubplot, 0.25, bottom=0)

subplot.set_title('Age against activity level')
subplot.set_xlabel('Age group')
subplot.set_xticks(ind + 0.25)
subplot.set_xticklabels(('16-30', '31-40', '41-50', '51-60', '61-70', '>70'))
subplot.set_ylabel('Activity level proportions within age group')
subplot.grid()

subplot.legend((inactive[0], active[0], veryActive[0]), ('Inactive', 'Active', 'Very active'))
subplot.autoscale_view()
```

In general, the chart shows that within each age group, activity levels are similar to those of the general population. The “61-70” and “70+” age groups have the highest proportion of people who are inactive, and the lowest proportion of people who are very active. The group with the highest proportion of people who are active is “31-40” and the highest proportion of people who are very active belongs to the “51-60” age group.



Based on the charts created, there do not seem to be significant differences between different segments of the population in terms of their lifestyle choices. The differences in lifestyle choices that do exist are very minor.

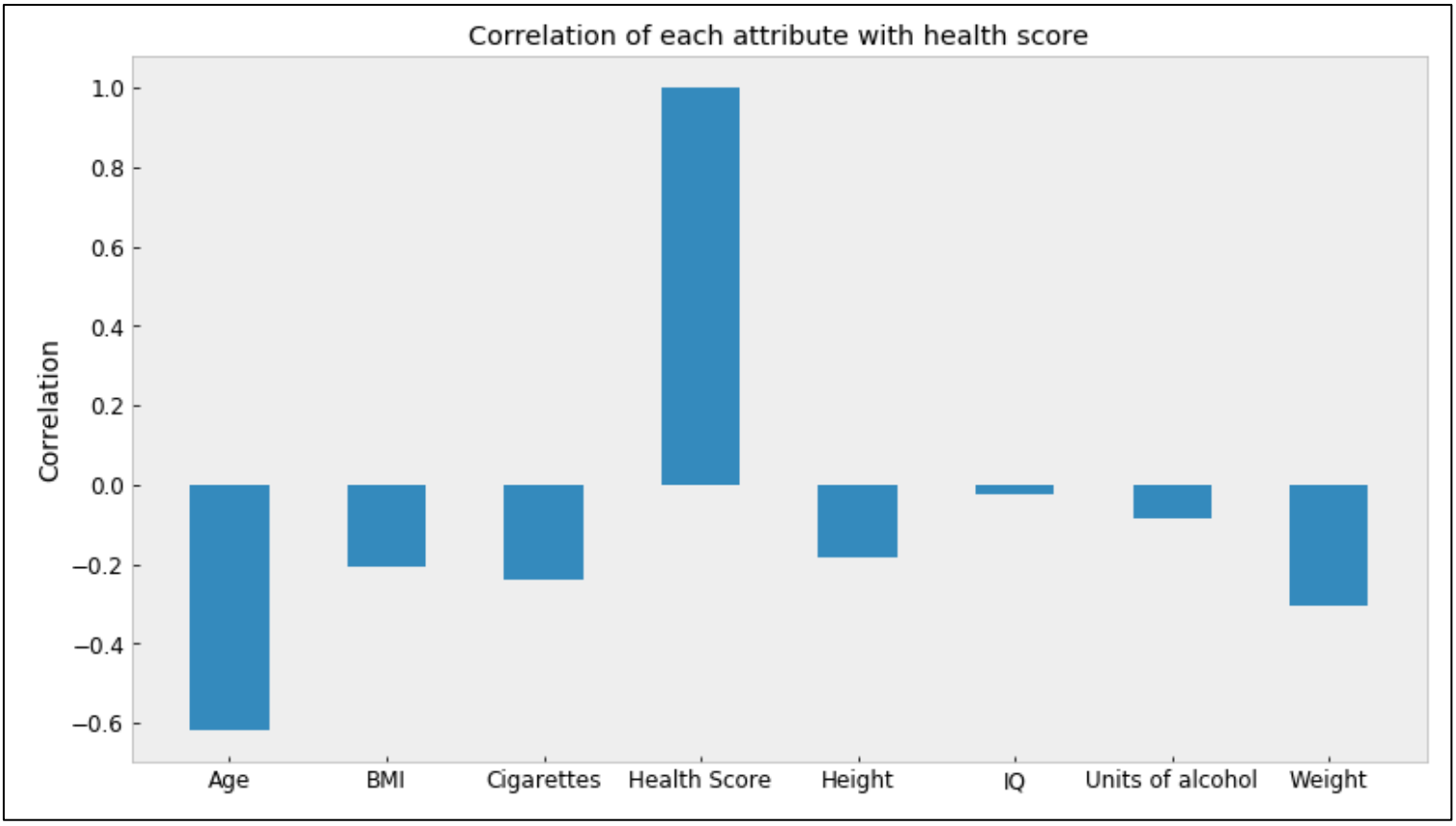
Which individual characteristics and lifestyle choices impact a person's health score, and to what extent?

In order to answer this question, a correlation matrix for the health score data has been calculated and plotted to measure the correlation between health score and each of the other features in the data set.

```
]: #Which individual characteristics and lifestyle choices impact a person's health scor

   corrValues = corr_matrix['Health Score (high is good)'].sort_values(ascending=False)
   corrValues
```

```
]: Health Score (high is good)    1.000000
   IQ                           -0.022800
   Units of alcohol per day     -0.086539
   Height in Inch               -0.184638
   BMI                          -0.206545
   Cigarettes per day           -0.241042
   Weight  in lbs               -0.305783
   Age                          -0.618599
   Name: Health Score (high is good), dtype: float64
```

```
]: plt.figure(figsize=(20, 10))
   plt.grid()
   plt.bar(x=['Health Score',
              'IQ',
              'Units of alcohol per day',
              'Height in Inch',
              'BMI',
              'Cigarettes per day',
              'Weight in lbs',
              'Age'],
           height=corrValues,
           width=0.5)
   plt.title('Correlation of each attribute with health score')
   plt.ylabel('Correlation')
```



Sex and activity level are not included in the correlation matrix because they are not numerical data. Of the features that are included in the correlation matrix, all of them have a negative correlation with health score, even if only a slight one. The three characteristics with the strongest impact are age, weight and cigarettes smoked. Health scores have a strong negative correlation with age. As age increases, health score decreases. The correlation is about -0.6; age is by far the most significant factor affecting health score. The second most significant factor affecting health score is weight, with a negative correlation of around -0.3. Number of cigarettes smoked is the next strongest correlation, around -0.22. This is followed by BMI, then height, then IQ.

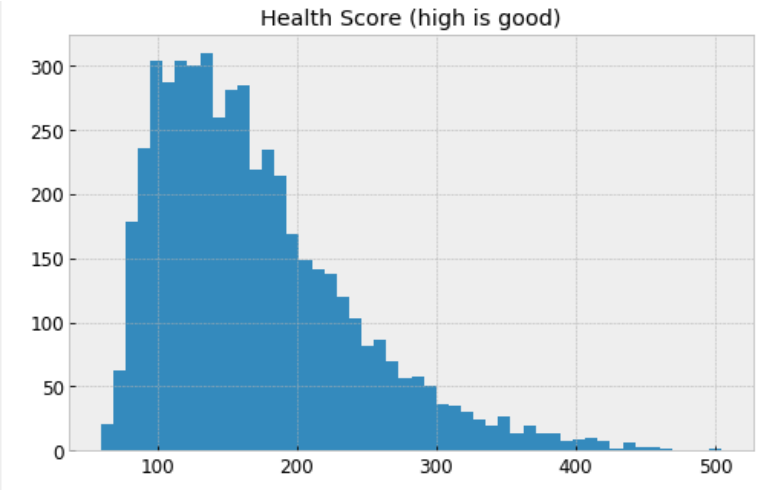
What would be the impact on the health scores of the overall population if nobody consumed alcohol or smoked?

In order to analyse this, a copy of the health score data has been made, with all smokers and drinkers removed. The two different versions of the health score data, one with smokers and drinkers, and one without, have been compared using histograms.

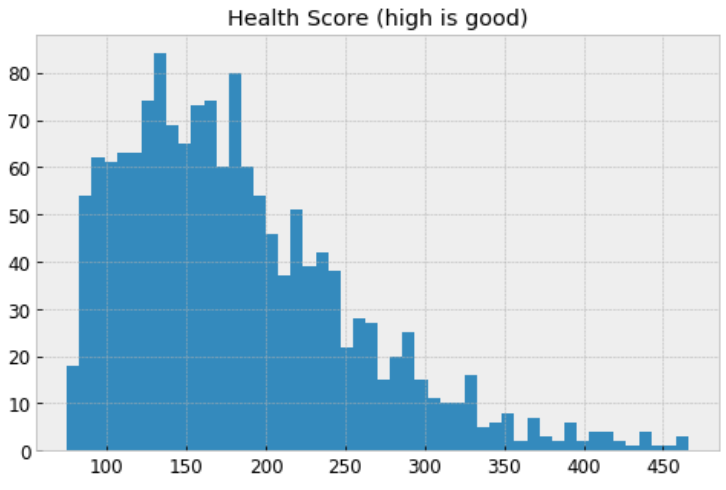
```
#What would be the impact on the overall population (in terms of health score) if nobody consumed alcoh

#create a copy of healthScores with all smokers removed
noSmokersOrDrinkers = healthScores.copy()
noSmokersOrDrinkers = noSmokersOrDrinkers.loc[noSmokersOrDrinkers['Units of alcohol per day'] == 0]
noSmokersOrDrinkers = noSmokersOrDrinkers.loc[noSmokersOrDrinkers['Cigarettes per day'] == 0]
```

```
healthScores.hist(bins=50, column='Health Score (high is good)', figsize=(8,5))
```

Comparing the distributions, it seems that the data set without smokers and drinkers has a higher proportion of people with high health scores (above 200). This impression is confirmed by the calculations shown below. 41.9% of people in the original data set have health scores above 200, compared to 48.7% of people in the set with smokers and drinkers removed.



Left: data set including smokers and drinkers



Right: data set with smokers and drinkers excluded

```
]: #Find proportions of people in each group who have health scores over 200. The most commonly occurring
   #so this helps show how many people have above-average scores in both the overall population, and the

   healthScoresUnder200 = healthScores.loc[healthScores['Health Score (high is good)'] <= 200]
   healthScoresOver200 = healthScores.loc[healthScores['Health Score (high is good)'] > 200]

   NSOD_Under200 = noSmokersOrDrinkers.loc[noSmokersOrDrinkers['Health Score (high is good)'] <= 200]
   NSOD_Over200 = noSmokersOrDrinkers.loc[noSmokersOrDrinkers['Health Score (high is good)'] > 200]

   HS_Under200Total = healthScoresUnder200['Health Score (high is good)'].sum()
   HS_Over200Total = healthScoresOver200['Health Score (high is good)'].sum()

   NSOD_Under200Total = NSOD_Under200['Health Score (high is good)'].sum()
   NSOD_Over200Total = NSOD_Over200['Health Score (high is good)'].sum()

   HS_PropOver200 = HS_Over200Total / (HS_Under200Total + HS_Over200Total)
   HS_PropOver200

]: 0.41905914908336217

]: NSOD_PropOver200 = NSOD_Over200Total / (NSOD_Under200Total + NSOD_Over200Total)
   NSOD_PropOver200

]: 0.4879659051872209
```

The original data set has an average health score of 170, whereas the data set without smokers and drinkers has an average health score of 183. This is a reasonably significant 13-point difference. It should be noted that this difference might not be entirely attributed to not smoking or drinking. For example, it may be the case that people who do not smoke or drink are also more conscious of their health, and are more likely to exercise and maintain a healthy weight.

```
In [21]: healthScores['Health Score (high is good)'].mean()
Out[21]: 170.02480496099218

In [22]: noSmokersOrDrinkers['Health Score (high is good)'].mean()
Out[22]: 183.0098231827112
```

Predicting health scores

(Note: the Python notebooks used for building these data mining models can be found within the ZIP file submitted.)

In order to predict the unknown health scores of the 20 individuals, a range of data mining models have been developed and applied. If entirely separate models using different techniques all give similar predictions, then the probability that the predictions are accurate is good. The models developed are a neural network model, a random forest model, a support vector machine model, and a linear regression model.

All four models have been created using the Scikit Learn data mining library (Python). The Scikit Learn library has a number of advantages: it is well-maintained and current, it has good documentation, and it has efficient and tested implementations of the various algorithms used to create each model. Scikit Learn also offers a consistent interface across each of its different model types.

Each of the models developed takes the same approach to pre-processing the data. The health scores data set is loaded into the program as described in the data analytics section of the report, and a column is added for BMI. After this, the textual data in the sex and exercise level columns is converted to numerical data as shown below.

```
#Convert columns containing non-numerical data into numerical data so they can be processed by the support vecto

#Sex: Male = 0, Female = 1
#Health Score (high is good): Inactive = 0, Active = 1, Very active = 2

healthScores['Sex'] = healthScores['Sex'].map({'Male': 0, 'Female': 1})
healthScores['Active'] = healthScores['Active'].map({'Inactive': 0, 'Active': 1, 'Very Active': 2})

healthScores.head()
```

	Age	Sex	Weight in lbs	Height in Inch	IQ	Units of alcohol per day	Cigarettes per day	Active	Health Score (high is good)	BMI
0	32	1	101	67	110	0	0	0	251	15.818661
1	47	1	138	67	84	2	0	1	193	21.613616
2	25	0	166	73	106	2	5	0	167	21.900817
3	75	0	184	73	99	2	0	0	93	24.275605
4	72	1	105	65	99	2	0	1	150	17.472721

After this, the data table is split into two parts. The first part consists of all of the columns (features) of the data minus the health scores column. This is used for input into the model. The second part consists of just the health scores column, and these are the values used to fit the model.

```
#create a new data frame (x) from the old one, but drop the health score column
#create a new data frame (y) from the health score column

x = healthScores.drop('Health Score (high is good)', axis=1)
y = healthScores['Health Score (high is good)']

y
```

```
0      251
1      193
2      167
3       93
4      150
5      120
6      167
7       90
8      181
```

The input and output data is then split into a training set and a test set. The training set is used to train and fit the model, and the test set is used to test the model against data it has not seen before, so that its accuracy can be evaluated. After being split into a training and test set, the input data is normalised by feature (column). This ensures that every feature in the data has an equal influence on the model.

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(x_train)
```

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)

x_train
```

```
array([[-0.04342542,  0.78926784, -0.46196499, ..., -0.42349441,
         0.42896596,  0.30123965],
       [ 0.01129791, -1.26699702,  1.24274569, ..., -0.42349441,
         0.42896596,  0.85786003],
       [ 1.21521117,  0.78926784, -1.13592038, ...,  1.64030455,
        -0.89902197, -1.7560743 ],
       ...,
       [-0.42648873, -1.26699702,  1.08416795, ...,  3.39453367,
         1.75695388,  0.89671341],
       [ 1.59827448, -1.26699702,  0.25163483, ..., -0.42349441,
        -0.89902197, -1.20349178],
       [-0.20759541, -1.26699702, -0.02587622, ..., -0.42349441,
         1.75695388, -0.35822402]])
```

Using the training data, the model is then trained to produce the correct outputs from any row of given input data, and if trained correctly, the model will be able to generalise so that it can predict suitable output values for any set of input data provided to it.

Random forest model

Decision trees are models built up of conditional branches or rules. Results are produced by performing a sequence of tests on the values of attributes. Random forests are a type of ensemble learning based on building a set of independent decision trees, getting results from each decision tree, and combining them to produce a final output. Random forests can be used for either classification or prediction / regression. The only thing that differs between these is how the decision trees' results are combined. For regression, the mean result is taken, whereas for classification, the mode result is taken.

Random forests address one of the main weaknesses of decision trees, which is overfitting to the training data. Because decision trees are based on a sequence of tests on attributes, a decision based on a bad rule is likely to have a knock-on effect on tests that come after it, and on the results of the model. Random forests avoid this by building many separate trees, each with different branch structures, and taking a consensus of their results.

The code below is used to fit a random forest to the training data.

```
RFModel = RandomForestRegressor(n_jobs=2, random_state=0)

RFModel.fit(train[features], y)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=2,
           oob_score=False, random_state=0, verbose=0, warm_start=False)
```

Once trained, the model is tested by making predictions on the test set. The predictions are then compared to the actual values to see how accurate the model is. The model gets 95.6% of its results correct (within a tolerance: the result must be within the range “actual health score ± 10% of max health score” to be deemed accurate).

```
preds = RFModel.predict(test[features])
```

```
preds
```

```
array([0.05313901, 0.27713004, 0.30426009, ..., 0.25717489, 0.16524664,
       0.45627803])
```

```
#pd.crosstab(test['Health Score (high is good)'], preds, rownames=['Actual health score'], colnames=
```

```
#Determine number of accurate predictions - a prediction is deemed to be accurate if it is within 10

actualValues = test['Health Score (high is good)'].values
totalNumValues = len(test)

numAccurateResults = 0

for i in range(0, len(preds)):
    if abs(preds[i] - actualValues[i]) < (0.1 * healthScores['Health Score (high is good)'].max()):
        numAccurateResults += 1

percentAccurateResults = (numAccurateResults / totalNumValues) * 100
percentAccurateResults
```

```
95.63106796116504
```

Once tested, the model is used to make predictions for the 20 individuals with unknown health scores as shown below.

```
#Perform the predictions for the 20 individuals with unknown health scores, and un-normalise the results

actualPredictions = RFModel.predict(population[features])

#Formulae for normalising and un-normalising values:
#normalisedVal = (x - min) / (max - min)
#x = (normalisedVal * (max - min)) + min

for i in range(0, len(actualPredictions)):
    actualPredictions[i] = (actualPredictions[i] * (maxHealthScore - minHealthScore)) + minHealthScore

actualPredictions
```

```
array([125.2, 238.8, 149.5, 244.1, 153.9,  99.2, 212.1, 150.2, 185.6,
       237.4,  87.7, 166.9, 113.4, 368.8, 279.1, 133.3, 154.2,  64.9,
       105. , 313.2])
```

```
population['Predicted health scores'] = actualPredictions
population
```

Neural network model

Neural networks are a commonly used and well-known machine learning / data mining technique. Neural networks are loosely based on the way a biological brain works. They have multiple nodes / neurons, each of which is connected to multiple other nodes. Nodes are arranged into three (or more) layers, the input layer, hidden layer(s) and output layer. Each node in a layer is connected to all of the nodes in the immediately neighbouring layers. For regression, the output layer may consist of a single node that, having received inputs from the nodes in the previous layer, will give a single value as output.

In order for a neural network to be useful, it must be trained. This is done by repeatedly providing it with a set of input values. These input values are fed through the nodes in each layer to the nodes in the subsequent layer. When a node receives a set of inputs, it will apply “weights” to each of them. It will also apply a function to the weighted inputs, and pass on the output to each of the nodes it is connected to in the next layer. Each time the neural network produces an output, a correction process is carried out. This involves a slight modification to the node weights in order to make the output slightly more similar to the desired output. Once trained, the neural network can be applied to data it has not seen before in order to make predictions.

One of the advantages of neural networks is their ability to learn complex and non-linear patterns or relationships. Another advantage is their ability to generalise from a limited set of training data so that they can produce accurate results for inputs they have not seen before (provided they are trained well).

The code below is used to fit a neural network to the health scores training data, and then perform health score predictions on the test set. The neural network has one hidden layer with 100 nodes, and has a maximum number of training iterations of 1500.

```
#1 hidden layer with 100 nodes, maximum training iterations allowed = 1500
mlp = MLPRegressor(hidden_layer_sizes=(100, ), max_iter=1500)
mlp.fit(x_train, y_train)
```

```
MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=1500, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
```

```
predictions = mlp.predict(x_test)
```

Next, the predictions made on the test set are compared to the actual health score values. The model gets 99.44% of its results correct (within a 10% tolerance of the actual health score).

```
predictions
```

```
array([242.49641048, 115.82130058,  80.08368756, ..., 118.18776187,
       103.19073626, 117.14706824])
```

```
#Determine number of accurate predictions - a prediction is deemed to be accurate if it is within 10% of t

actualValues = y_test.values
totalNumValues = len(y_test)

numAccurateResults = 0

for i in range(0, len(predictions)):
    if abs(predictions[i] - actualValues[i]) < (0.1 * healthScores['Health Score (high is good)'].max()):
        numAccurateResults += 1

percentAccurateResults = (numAccurateResults / totalNumValues) * 100
percentAccurateResults
```

```
99.44
```

Having verified the model's accuracy, the last step is to have the model predict health score values for the 20 individuals with unknown health scores as shown below.

```
#Perform the predictions for the 20 individuals with unknown health scores

actualPredictions = mlp.predict(inputs)
actualPredictions
```

```
array([122.15031849, 241.66058012, 134.56091708, 257.33488732,
       161.3497996 , 105.83044711, 206.33660161, 143.542603  ,
       187.54844253, 240.78067458,  96.43946583, 181.24663692,
       115.13143664, 364.43637397, 298.99245853, 146.90654987,
       154.52892333,  75.65797156, 118.95438172, 309.13987183])
```

```
population['Predicted health scores'] = actualPredictions
population
```

Linear regression model

Linear regression is a very simple type of model; it seeks to find a straight line of best fit for the data. It will take in all of the data features as input and find the equation of the line that best maps these to the correct output values. In order to find this line, linear regression methods often use the method of least squares. This means that the model will seek to minimise the sum of the squared errors of each data point from the line. Squared error is used so that positive and negative error numbers will not cancel each other out. By minimising the sum of squared errors, the line will eventually be placed in an optimal position. Using this line, predictions can be made by projecting input values upwards on to the line from the x-axis, and finding the y-coordinate where they intersect.

One of the advantages of linear regression is its simplicity. It is very easy to program and understand. It is also one of the best models for data sets which have a linear relationship between their input and output data. The downside of this is that linear regression is not well suited to problems with non-linear relationships between variables (e.g. curved lines). With problems like this, linear regression is not very good at representing the underlying distribution of data. For example, outliers can have a significant impact on how well the line fits the majority of the data points. Also, some sets of data points may be different in layout and structure, but mathematically may have the same line of best fit.

The code below is used to fit the linear regression model to the health scores training data, and then perform health score predictions on the test set.

```
reg = linear_model.LinearRegression()
reg.fit(x_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
predictions = reg.predict(x_test)
predictions
```

```
array([223.44322589, 164.06040589,  83.36810597, ..., 122.12792441,
        69.88082905, 229.28610678])
```

Next, the predictions made on the test set are compared to the actual health score values. The model gets 91.52% of its results correct (within a 10% tolerance of the actual health score).

```
#Determine number of accurate predictions - a prediction is deemed to be accurate if it is within 10% of th

actualValues = y_test.values
totalNumValues = len(y_test)

numAccurateResults = 0

for i in range(0, len(predictions)):
    if abs(predictions[i] - actualValues[i]) < (0.1 * healthScores['Health Score (high is good)'].max()):
        numAccurateResults += 1

percentAccurateResults = (numAccurateResults / totalNumValues) * 100
percentAccurateResults
```

```
91.52
```

Having verified the model's accuracy, the last step is to have the model predict health score values for the 20 individuals with unknown health scores as shown below.

```
#Perform the predictions for the 20 individuals with unknown health scores

actualPredictions = reg.predict(inputs)
actualPredictions
```

```
array([120.9802755 , 235.13030534, 160.24533515, 238.95513606,
       157.54189   , 106.49477209, 209.6687886 , 152.34992402,
       197.802032  , 255.40433042,  89.4057399 , 182.69671924,
       126.60536721, 286.92627916, 280.09822621, 129.81112645,
       137.61711444,  28.46212125, 151.507361  , 260.16370232])
```

```
population['Predicted health scores'] = actualPredictions
population
```

Support vector machine model

A support vector machine is a model that stores data entries as points in an n-dimensional space (n = the number of features in the data set). A support vector machine is provided with training data, and all of the data entries provided must fall into one of two categories. Based on these data entries, a support vector machine seeks to find a hyperplane (essentially a line mapped to the n-dimensional space) that will optimally separate the data points belonging to different categories. An optimal hyperplane is one where all of the data points belonging to each category are as far away from the hyperplane as possible. Support vector machines are normally used for classification, but can be used for regression by altering the way the hyperplane is constructed.

Support vector machines can be used for linear or non-linear problems. The hyperplane is normally a straight line, but non-linear problems can still be modelled effectively. This is done by mathematically transforming the non-linear problem into a linear problem by mapping the data points into a higher-dimensional space. The SVM will build the hyperplane in the higher-dimensional space as a straight line, and when mapped back down into n-dimensions, it will take a curved form that effectively separates the two categories.

The main advantages of SVMs are their effectiveness in high-dimensional spaces (even when they are given more features than samples), and their performance efficiency (they only need to consider the data points closest to the hyperplane, most other data points can be safely ignored).

The code below trains an SVM regression model using the health scores data. It then performs health score predictions on the test set. It also analyses the predictions to determine their accuracy. The model gets 93.68% of its results correct (within a 10% tolerance of the actual health score).

```
from sklearn import svm

SVMModel = svm.SVR()
SVMModel.fit(x_train, y_train)
```

```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='auto',
  kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

```
predictions = SVMModel.predict(x_test)
predictions
```

```
array([230.68037826, 177.50009328, 115.81743229, ..., 171.91786399,
       137.6676482 , 133.33995711])
```

```
#Determine number of accurate predictions - a prediction is deemed to be accurate if it is within 10% of t

actualValues = y_test.values
totalNumValues = len(y_test)

numAccurateResults = 0

for i in range(0, len(predictions)):
    if abs(predictions[i] - actualValues[i]) < (0.1 * healthScores['Health Score (high is good)'].max()):
        numAccurateResults += 1

percentAccurateResults = (numAccurateResults / totalNumValues) * 100
percentAccurateResults
```

```
93.67999999999999
```

After determining that the model's results are accurate, it is used to make health score predictions for the 20 individuals with unknown health scores as shown below.

```
#Perform the predictions for the 20 individuals with unknown hea

actualPredictions = SVMModel.predict(inputs)
actualPredictions
```

```
array([123.35231166, 226.01390073, 144.72007   , 258.33212874,
       154.02667829, 121.96298963, 218.55027763, 147.52397929,
       175.09065003, 233.43121855, 110.5132166 , 172.59642679,
       115.52829427, 283.85941616, 251.91401839, 134.87874442,
       150.09005182,  77.60640497, 131.35102962, 286.1040856 ])
```

```
population['Predicted health scores'] = actualPredictions
population
```

Predictions

All of the models were trained using the majority of the data (approximately 3500 records), and then tested on a test set of unseen data (approximately 1500 records). The health score predictions made by each model were then compared to the true health score values. All four models were able to

obtain fairly good accuracy rates. In each model, upwards of 90% of the predictions were within 10% of the maximum health score away from the true health score values.

The neural network model was particularly accurate, being within 10% of the true health score value up to 99% of the time. The neural network had the highest accuracy (99.4%), followed by the random forest (95.6%), followed by the support vector machine (93.7%), and perhaps unsurprisingly, the least accurate model was the linear regression (91.5%).

All of the models made similar health score predictions for most of the 20 individuals with unknown health scores. There were only a few cases where the predictions differed significantly. Therefore, the method used to calculate the final answer is simply to take the mean of the results for each individual from each of the four models.

Person ID	Random forest model	Neural network model	Linear regression model	Support vector machine model	My final answer is…
1	125.2	122.150318	120.980275	123.352312	122.9207263
2	238.8	241.66058	235.130305	226.013901	235.4011965
3	149.5	134.560917	160.245335	144.72007	147.2565805
4	244.1	257.334887	238.955136	258.332129	249.680538
5	153.9	161.3498	157.54189	154.026678	156.704592
6	99.2	105.830447	106.494772	121.96299	108.3720523
7	212.1	206.336602	209.668789	218.550278	211.6639173
8	150.2	143.542603	152.349924	147.523979	148.4041265
9	185.6	187.548443	197.802032	175.09065	186.5102813
10	237.4	240.780675	255.40433	233.431219	241.754056
11	87.7	96.439466	89.40574	110.513217	96.01460575
12	166.9	181.246637	182.696719	172.596427	175.8599458
13	113.4	115.131437	126.605367	115.528294	117.6662745
14	368.8	364.436374	286.926279	283.859416	326.0055173
15	279.1	298.992459	280.098226	251.914018	277.5261758
16	133.3	146.90655	129.811126	134.878744	136.224105
17	154.2	154.528923	137.617114	150.090052	149.1090223
18	64.9	75.657972	28.462121	77.606405	61.6566245
19	105	118.954382	151.507361	131.35103	126.7031933
20	313.2	309.139872	260.163702	286.104086	292.151915

Findings and recommendations

With regard to the analysis of data, no significant differences were found between different segments of the population in terms of their lifestyle choices. Both sexes make almost exactly the same lifestyle choices in terms of alcohol consumption, cigarette use and exercise level. The same goes for each different age group. There is no correlation between age and the amount of alcohol consumption, cigarette use or exercise taken.

It was found that by far the most significant characteristic that negatively affects an individuals' health score is their age, which is obviously not within their control. The second two most significant characteristics were weight and cigarettes smoked (both of which also impact health score negatively). If the NHS wished to improve patient health scores, then their efforts would be best focused on encouraging people to lose weight, and stop smoking.

It was also found that removing those who smoke and drink from the data set was able to increase the average health score from 170 to 183, an increase of 13 points. This is a modest increase, considering that the health score values range from around 60 to around 500. However, it is still worth pursuing, by encouraging people to reduce the amount they drink, and stop smoking.

The data mining models built to predict health scores were all able to achieve broad accuracy. The predictions were almost all accurate to ±10% of the maximum health score found within the data. All of the models would be effective in their current state for producing broad predictions for health scores, but none of them could yet address the more difficult problem of making predictions with precision. There is plenty of potential for optimising the models in future, since this work has focused more on implementing a range of models to assess their viability, rather than optimisation of specific models.