Yay or Nay? Using Machine Learning to Predict your Marathon Success.

In this assignment, I will use supervised learning to predict whether or not a runner can successfully finish a marathon. This problem is a binary classification with the target variabler *"Failure"* expected to be [0,1] in case of success/insuccess in finishing the race. The training dataset contains a list of 3000 marathon finishers, it is 15-dimensional, consisting in both numerical and categorical variables.

First let's import the Python libraries.

In [1]:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pylab as pl
import statistics as st

from pandas.tools.plotting import scatter_matrix
from scipy import stats
from matplotlib import cm

from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import RFE
from sklearn.preprocessing import Imputer
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVR
from sklearn.svm import SVC
```

Here, I open the .csv file (stored in the current working directory) with *Pandas*, create a *DataFrame* structure and print the top five elements of the dataset.

In [2]:

```
# Importing the dataset located in the current working directory
dataset = pd.read_csv('Marathon_3000.csv')
dataset.head()
```

Out[2]:

	BMI	AGE	PHYSICAL STATUS	WEATHER CONDITION	FAINT RATE	SLOPE	SHOES TYPE	SEX	ELEVATION FACTOR	MONTH	WATER INTAKE	EXPERIENCED	BLO SUG
0	17.990	26	1	0.16	0.01	-8.12	0	1	332.5	8	0	1	5.13
1	20.704	36	1	0.30	0.01	-4.52	2	0	328.0	11	1	0	6.15
2	19.156	34	1	0.30	0.01	-1.08	0	0	247.0	0	1	0	6.36
3	16.802	35	1	0.19	0.02	7.44	1	0	408.0	7	3	1	6.62
4	17.140	23	2	0.39	0.01	30.52	0	1	308.0	2	2	0	6.15

In [3]:

```
print(dataset.shape)
print('Original dataset')
```

```
(3000, 16)
Original dataset
```

The Dataset is 3000x16, 15 explicative variables (featureS) plus the *'Failure'* target variable.

With this command I am going to store the feature names in a list for later use.

In [4]:

```
# Features & Target variable names
features = (list(dataset.columns.values))[0:15]
variable = 'FAILURE'
```

Data Exploration

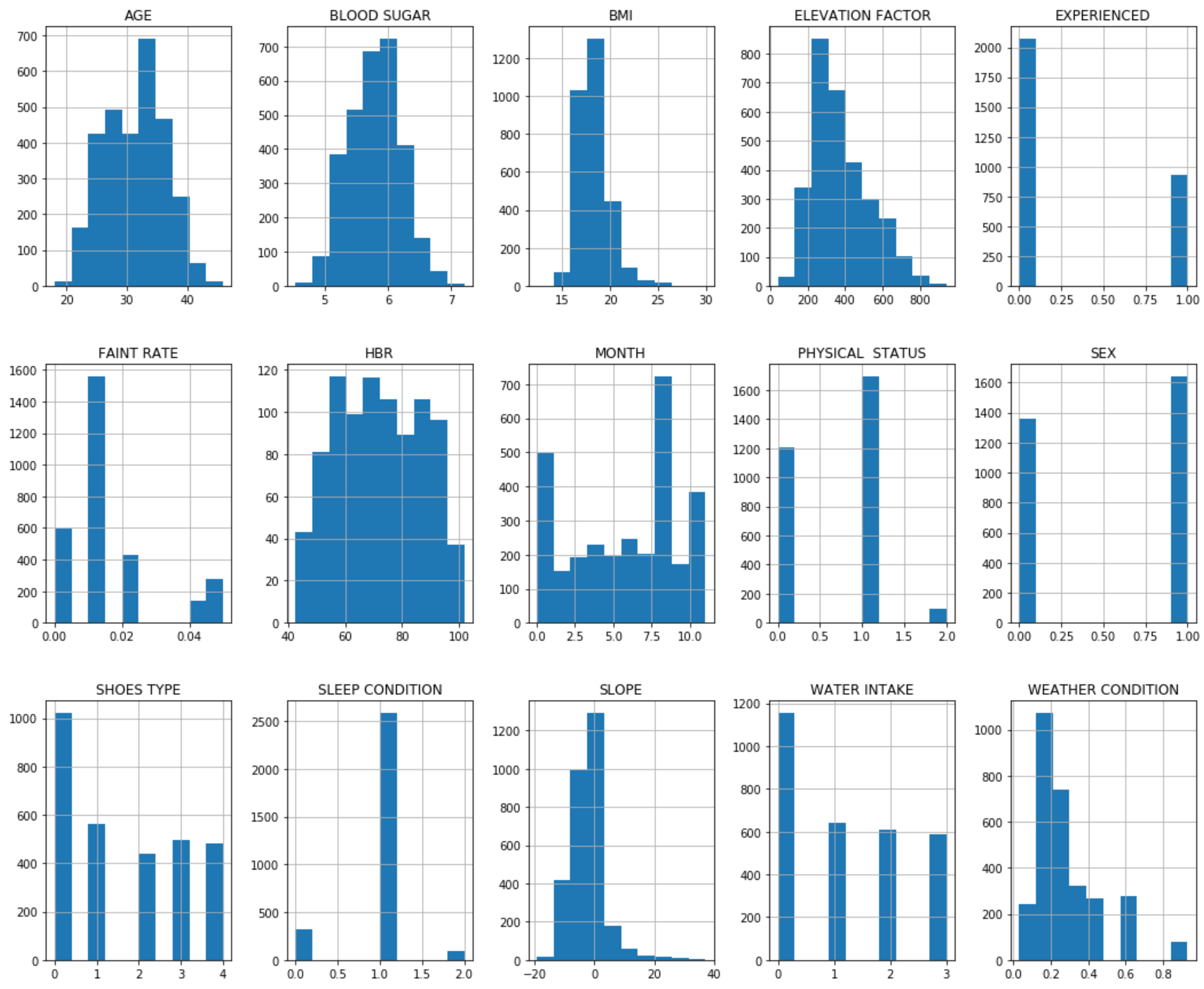
We can se that the proportion of *"Failure"* cases is 66% : only one third of the participants arrives to ending point.

The histograms tells us more about the feature frequency distribution. In particular the variables *"age", "blood sugar", "hbr"* and *"month"* show a gaussian-like distribution.
Some variables are left-skewed, such as *"slope", "elevation factor" and "bmi"*. Moreover, the majority of the *"bmi"* cases falls in the range 15-20 which tells us that the most of the participants are in well shape.
Almost the two thirds of the runners are experienced, and there is no big difference between male and female.

In [5]:

```
# Explorative histograms
print('\n',variable, "PERCENTAGE :", round(1-sum(dataset['FAILURE']/len(dataset)),3))
dataset.drop('FAILURE', axis=1).hist(figsize=(18,15), layout=(3,5))
plt.show()
```

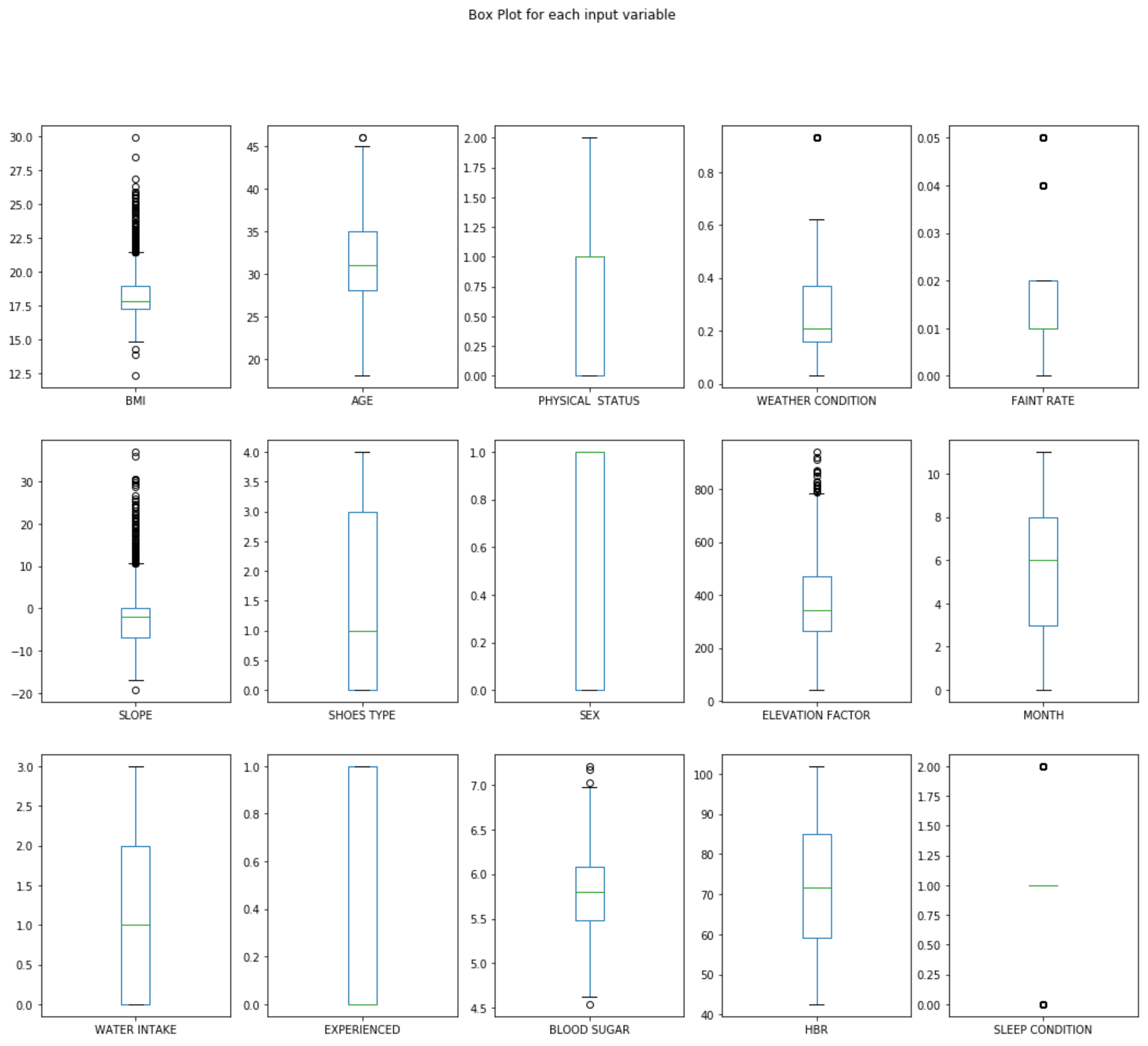
```
 FAILURE PERCENTAGE : 0.664
```



The boxplots confirms the first insight about the gaussian distribution of some variables, but reveals the presence of outliers in the *"bmi"," weather condition", "slope", "elevation factor", "hbr" and "sleep condition"*.

In [6]:

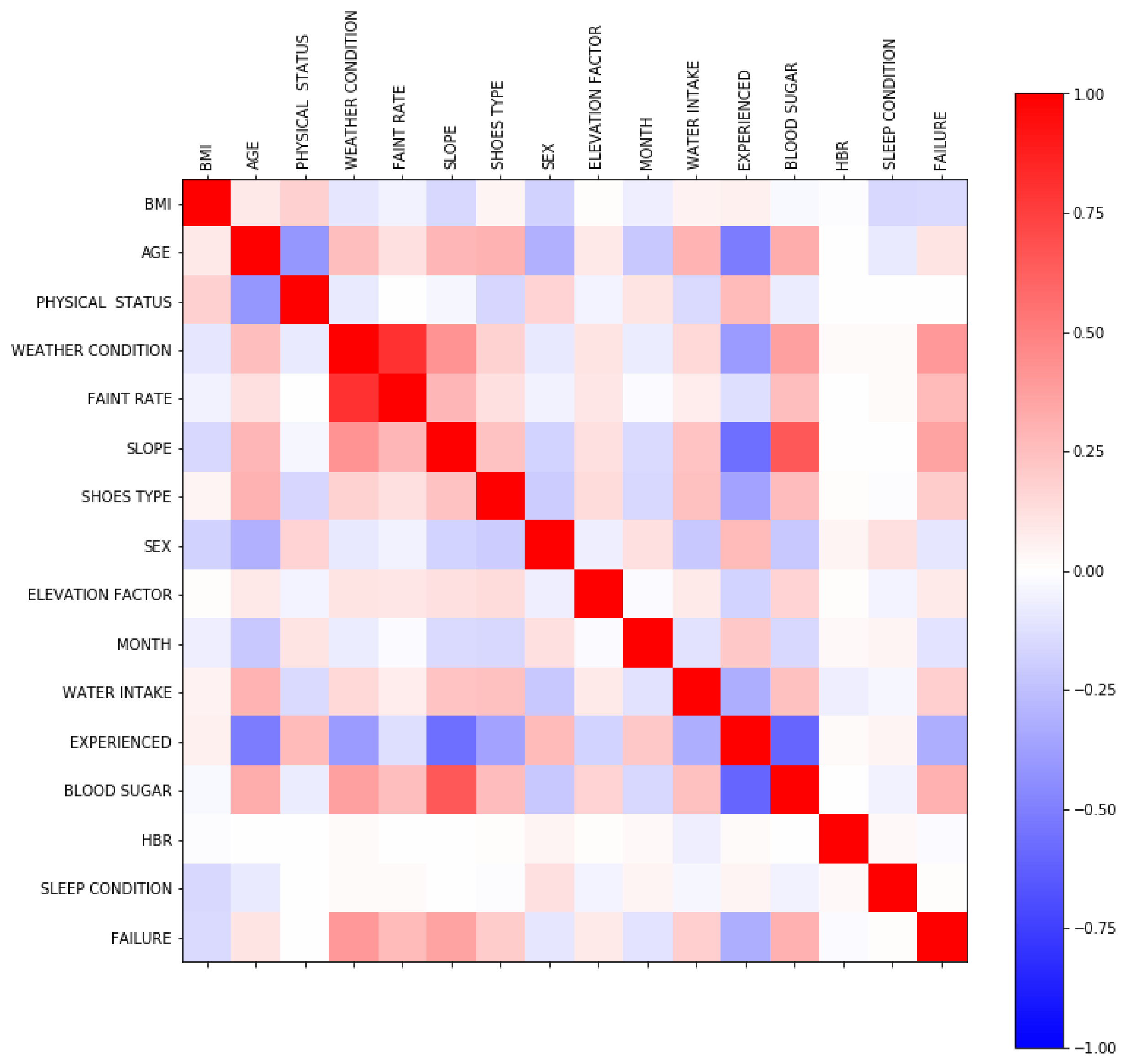
```
# Explorative analysis : boxplots
dataset.drop('FAILURE', axis=1).plot(kind='box', subplots=True, layout=(3,5), sharex=False, sharey=
False, figsize=(18,15),
                                                    title='Box Plot for each input variable')
plt.savefig('dataset_bp')
plt.show()
```



The correlation plot shows that *'hbr'* seems less correlated to the other variables, but also a strong correlation between the fialure and the track condition, slope and weather.
Since there are categorical variables, I am going to create single dummies to understand if there is a correlation a particular category.

In [7]:

```
# Plot correlation matrix
correlations = dataset.corr()

fig = plt.figure(figsize = (12,12))
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin=-1, vmax=1, cmap = 'bwr')
fig.colorbar(cax)
ticks = np.arange(0,16,1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(list(dataset.columns.values),rotation='vertical')
ax.set_yticklabels(list(dataset.columns.values) )

plt.show()
```



Data Pre-Processing : Missing Values & Outliers

The statistical summary reveals missing values in the *"hbr"* variable.
I am going to impute those *NaN* with the *median*, since there is presence of outliers in this particular distribution which can affect the mean value.

In [8]:

```
# Data Preprocessing step 1
# Original dataset stats description

# print(dataset.describe())
for var in features:
    print('{}'.format(var) + ' :\n ' + str(stats.describe(dataset[var])))
    # some NaN in HBR
```

```
BMI :
 DescribeResult(nobs=3000, minmax=(12.312000000000001, 29.932), mean=18.184712,
variance=2.51918040118973, skewness=1.4610051663068762, kurtosis=4.311567248162681)
AGE :
 DescribeResult(nobs=3000, minmax=(18, 46), mean=31.249666666666666, variance=24.337445704123592,
skewness=0.03462749112267123, kurtosis=-0.5433784800785633)
PHYSICAL  STATUS :
 DescribeResult(nobs=3000, minmax=(0, 2), mean=0.6313333333333333, variance=0.29885117261309324, s
kewness=0.07263010470335088, kurtosis=-0.8706019790673154)
WEATHER CONDITION :
 DescribeResult(nobs=3000, minmax=(0.03, 0.93), mean=0.2829866666666666,
variance=0.03358140695787485, skewness=1.5293124184171656, kurtosis=2.54084704902958)
FAINT RATE :
 DescribeResult(nobs=3000, minmax=(0.0, 0.05), mean=0.014550000000000002,
variance=0.00020339863287762594, skewness=1.4669827591221964, kurtosis=1.226723875616659)
```

```
SLOPE :
 DescribeResult(nobs=3000, minmax=(-19.28, 37.0), mean=-2.37536, variance=32.17366729283094,
skewness=1.5329078751356202, kurtosis=6.0492320824046)
SHOES TYPE :
 DescribeResult(nobs=3000, minmax=(0, 4), mean=1.618, variance=2.212813604534845,
skewness=0.3443073534523384, kurtosis=-1.335098966722179)
SEX :
 DescribeResult(nobs=3000, minmax=(0, 1), mean=0.5476666666666666, variance=0.24781049238635097, s
kewness=-0.19153904934575358, kurtosis=-1.963312792575725)
ELEVATION FACTOR :
 DescribeResult(nobs=3000, minmax=(41.5, 939.5), mean=376.184, variance=22846.280404134715,
skewness=0.7394505708552247, kurtosis=-0.022264186111953244)
MONTH :
 DescribeResult(nobs=3000, minmax=(0, 11), mean=5.7796666666666665, variance=11.015458375013893, s
kewness=-0.2910892668536668, kurtosis=-1.0750047743426516)
WATER INTAKE :
 DescribeResult(nobs=3000, minmax=(0, 3), mean=1.212, variance=1.3308329443147717,
skewness=0.34952279340284, kurtosis=-1.350372161628967)
EXPERIENCED :
 DescribeResult(nobs=3000, minmax=(0, 1), mean=0.30966666666666665, variance=0.21384450372346336,
skewness=0.8233191129576352, kurtosis=-1.3221456382386527)
BLOOD SUGAR :
 DescribeResult(nobs=3000, minmax=(4.54, 7.21), mean=5.79151, variance=0.16836274081360453,
skewness=0.053906103352045646, kurtosis=-0.3143836388238763)
HBR :
 DescribeResult(nobs=3000, minmax=(nan, nan), mean=nan, variance=nan, skewness=nan, kurtosis=nan)
SLEEP CONDITION :
 DescribeResult(nobs=3000, minmax=(0, 2), mean=0.922, variance=0.1326268756252084, skewness=-
0.9632369613766634, kurtosis=3.7857776778359575)
```

In [9]:

```
# Transforming missing data in HBR using 'median'
dataset['HBR'].isnull().sum() # 2110 nan
dataset['HBR'].fillna(dataset['HBR'].median(), inplace=True)
# it took place
dataset.head()
```

Out[9]:

	BMI	AGE	PHYSICAL STATUS	WEATHER CONDITION	FAINT RATE	SLOPE	SHOES TYPE	SEX	ELEVATION FACTOR	MONTH	WATER INTAKE	EXPERIENCED	BLO SUG
0	17.990	26	1	0.16	0.01	-8.12	0	1	332.5	8	0	1	5.13
1	20.704	36	1	0.30	0.01	-4.52	2	0	328.0	11	1	0	6.15
2	19.156	34	1	0.30	0.01	-1.08	0	0	247.0	0	1	0	6.36
3	16.802	35	1	0.19	0.02	7.44	1	0	408.0	7	3	1	6.62
4	17.140	23	2	0.39	0.01	30.52	0	1	308.0	2	2	0	6.15

Now, let's handle the outliers. First, let's count how many values exceed 3 standard deviations from the mean for each feature.

In [10]:

```
# detect the values which are over 3 standard deviations from the mean.
for name in features:
    print(name, " outliers : ", len(dataset[np.abs(dataset[name]-dataset[name].mean()) > (3*dataset
[name].std())]))
```

```
BMI  outliers :  43
AGE  outliers :  0
PHYSICAL  STATUS  outliers :  0
WEATHER CONDITION  outliers :  81
FAINT RATE  outliers :  0
SLOPE  outliers :  48
SHOES TYPE  outliers :  0
SEX  outliers :  0
ELEVATION FACTOR  outliers :  7
MONTH  outliers :  0
WATER INTAKE  outliers :  0
```

```
EXPERIENCED  outliers :  0
BLOOD SUGAR  outliers :  4
HBR  outliers :  70
SLEEP CONDITION  outliers :  0
```

The incidence of outliers is pretty significant and applying a machine learning model with those data would bias the prediction.
I'am going to remove the outliers using the *'Z-score'*.

In order to not reduce the dataset too much I am going to set the threshold to 3.5 std.
For each column of the dataset, first it computes the Z-score of each value, relative to the column mean and standard deviation. Then is taken the absolute of Z-score because the direction does not matter, only if it is below the threshold of 3.5 standard deviation. The method *.all(axis=1)* ensures that for each row, all column satisfy the constraint.

After removing the outliers, the dataset is significantly reduced in length (2842X16).

In [11]:

```
dataset = dataset[(np.abs(stats.zscore(dataset)) < 3.5).all(axis=1)]
print(dataset.shape)
print('Cleaned data')
```

```
(2845, 16)
Cleaned data
```

Data Preprocessing : Encoding categorical Variables & Create dummies

Encoding categorical data and binarize the target is an important step to follow before pass to the next steps.
Hence, I am going to use the pandas method *.get_dummies* to transform the categorical features into dummy variables.

In [12]:

```
# Data Preprocessing
# One Hot Encoding with pandas

# categories
categories = ['PHYSICAL  STATUS', 'SHOES TYPE', 'SEX', 'WATER INTAKE', 'EXPERIENCED', 'SLEEP CONDIT
ION', 'MONTH']

# Convert categorical variables into dummy variables
d_dataset = pd.get_dummies(dataset, columns = categories)

# New dataset dummies: 39 features
d_features = list(d_dataset.drop('FAILURE', axis = 1).columns.values)

print('List of new features :\n',d_features)
print('\nDataset shape :',d_dataset.shape)
print('\nNumber of features:',len(d_features))
d_dataset.head()
```

```
List of new features :
 ['BMI', 'AGE', 'WEATHER CONDITION', 'FAINT RATE', 'SLOPE', 'ELEVATION FACTOR', 'BLOOD SUGAR', 'HB
R', 'PHYSICAL  STATUS_0', 'PHYSICAL  STATUS_1', 'PHYSICAL  STATUS_2', 'SHOES TYPE_0', 'SHOES
TYPE_1', 'SHOES TYPE_2', 'SHOES TYPE_3', 'SHOES TYPE_4', 'SEX_0', 'SEX_1', 'WATER INTAKE_0', 'WATE
R INTAKE_1', 'WATER INTAKE_2', 'WATER INTAKE_3', 'EXPERIENCED_0', 'EXPERIENCED_1', 'SLEEP
CONDITION_0', 'SLEEP CONDITION_1', 'SLEEP CONDITION_2', 'MONTH_0', 'MONTH_1', 'MONTH_2',
'MONTH_3', 'MONTH_4', 'MONTH_5', 'MONTH_6', 'MONTH_7', 'MONTH_8', 'MONTH_9', 'MONTH_10',
'MONTH_11']

Dataset shape : (2845, 40)

Number of features: 39
```

Out[12]:

	BMI	AGE	WEATHER CONDITION	FAINT RATE	SLOPE	ELEVATION FACTOR	BLOOD SUGAR	HBR	FAILURE	PHYSICAL STATUS_0	...	MONTH_2	MONTH_3	M
0	17.990	26	0.16	0.01	-8.12	332.5	5.13	55.00	0	0	...	0	0	0
1	20.704	36	0.30	0.01	-4.52	328.0	6.15	71.65	0	0	...	0	0	0

	BMI	AGE	WEATHER CONDITION	FAINT RATE	SLOPE	ELEVATION FACTOR	BLOOD SUGAR	HBR	FAILURE	PHYSICAL STATUS_0	...	MONTH_2	MONTH_3	M
2	19.[illegible]	34	0.30	0.01	-1.08	247.0	6.36	71.[illegible]	0	0	...	0	0	0
3	16.802	35	0.19	0.02	7.44	408.0	6.62	54.50	0	0	...	0	0	0
5	20.042	38	0.04	0.01	-0.20	160.5	5.96	71.65	0	1	...	0	0	0

5 rows × 40 columns

At this point is useful to group all the features in a X feature matrix and the *'Failure'* target in a y (binary) vector.

In [13]:

```
# Divide the X matrix and target vector
X = d_dataset.drop('FAILURE', axis = 1)
y = d_dataset['FAILURE']

# Binarize the y
labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)
```

Training set and Test set

Since the numerosity of the dataset is quite small, I am going to use 80% of the observation for the training set and remaining 20% for the test set.
Before applying the model, I am going to scale the data with the StandardScaler of scikitlearn. The StandardScaler applies the z-transformation, by subtracting the mean and dividing the standard deviation from each value.
Those are the numerosities:

In [14]:

```
# Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)

# Feature Scaling. After scaling data are converted into arrays
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)

print('Train set feature matrix : {}'.format(X_train.shape),
      '\nTrain set y vector : {}'.format(y_train.shape),
      '\nTest set feature matrix : {}'.format(X_test.shape),
      '\ntest set y vector : {}'.format(y_test.shape)
     )
```

```
Train set feature matrix : (2276, 39)
Train set y vector : (2276,)
Test set feature matrix : (569, 39)
test set y vector : (569,)
```

Machine Learning with Logistic Regression

The first function trains a logistic regression for a given X feature matrix (scaled) and a y target.
The perks of using the Logistic Regression model are:

```
- it's generalizable, doesn't overfit the training set data.
- it has a good predictive power.
- It is easy to interpret.
```

For evaluations criteria I am going to use F1 score and Accuracy, both measured on the test set.

Feature Selection

The Logistic Regression model doesn't need many features. In fact a lot of features can increase training overfitting and reduce the predictive power on the testing set. Here, I am going to use a *Recursive Feature Elimination* technique to rank the features according to their importance in the Logistic Regression model.
Then, I am going to optimize the number of features to include, in order to have the best predictive power.

In [29]:

```
# Feature Selection for Logistic Regression model
X = d_dataset.drop('FAILURE', axis = 1)
y = d_dataset['FAILURE']

# Recursive Feature Elimination
estimator = LogisticRegression()
selector = RFE(estimator, step = 1)
selector.fit(X, y)

# Ordering the most important variables for Logistic Regression model
print ("Features sorted by their rank:")
ranked_vars=(sorted(zip(map(lambda x: round(x, 4), selector.ranking_), d_features)))
print(ranked_vars)
```

```
Features sorted by their rank:
[(1, 'BLOOD SUGAR'), (1, 'BMI'), (1, 'EXPERIENCED_0'), (1, 'FAINT RATE'), (1, 'MONTH_11'), (1, 'MO
NTH_2'), (1, 'MONTH_3'), (1, 'MONTH_4'), (1, 'MONTH_7'), (1, 'MONTH_8'), (1, 'PHYSICAL  STATUS_0')
, (1, 'PHYSICAL  STATUS_2'), (1, 'SHOES TYPE_0'), (1, 'SHOES TYPE_2'), (1, 'SHOES TYPE_3'), (1, 'S
HOES TYPE_4'), (1, 'WATER INTAKE_0'), (1, 'WATER INTAKE_3'), (1, 'WEATHER CONDITION'), (2, 'MONTH_
1'), (3, 'MONTH_6'), (4, 'MONTH_0'), (5, 'PHYSICAL  STATUS_1'), (6, 'MONTH_5'), (7, 'MONTH_10'), (
8, 'SEX_1'), (9, 'SEX_0'), (10, 'WATER INTAKE_1'), (11, 'SLEEP CONDITION_2'), (12, 'WATER INTAKE_2
'), (13, 'SLOPE'), (14, 'SLEEP CONDITION_0'), (15, 'SLEEP CONDITION_1'), (16, 'EXPERIENCED_1'), (1
7, 'SHOES TYPE_1'), (18, 'AGE'), (19, 'MONTH_9'), (20, 'HBR'), (21, 'ELEVATION FACTOR')]
```

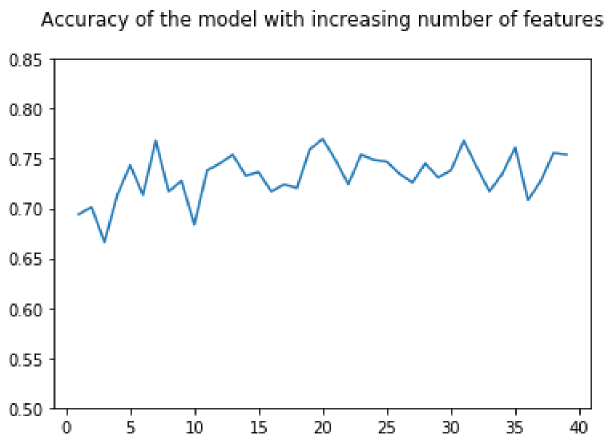
Here I am going to plot the accuracy and F1 scores of logistic regression model fit on our data.
The following loop is going to build/evaluate a logistic model with an increasing number of variables, starting from 1 variable.
Then I am plotting the Accuracy and F1 scores to see if there is an optimal number of features to include in the model.

In [30]:

```
# Feature Selection Finding the optimal number of features

numvar = []
accuracy = []
f1_score = []

for v in range(1,40):
    V = v
    for i in range(V):
        sel_features = [ranked_vars[i][1] for i in range(V)]
    # New X feature matrix with V selected variables
    X = d_dataset[sel_features]
    # Splitting the dataset into the Training set and Test set
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
    # Feature Scaling. After scaling data are converted into arrays
    sc_X = StandardScaler()
    X_train = sc_X.fit_transform(X_train)
    X_test = sc_X.transform(X_test)
    # Train Predict and print results
    logistic_class = LogisticRegression()
    logistic = logistic_class.fit(X_train, y_train)
    y_hat= logistic.predict(X_test)

    cm = confusion_matrix(y_test, y_hat)
    tn = cm[0,0]
    fn = cm[0,1]
    fp = cm[1,0]
    tp = cm[1,1]
    precision = tp/(tp+fp)
    recall = tp/(tp+fn)
    f1 = 2*((precision*recall)/(precision+recall))
    acc = np.sum(cm.diagonal()/np.sum(cm))

    numvar.append(i+1)
    accuracy.append(acc)
    f1_score.append(f1)
```

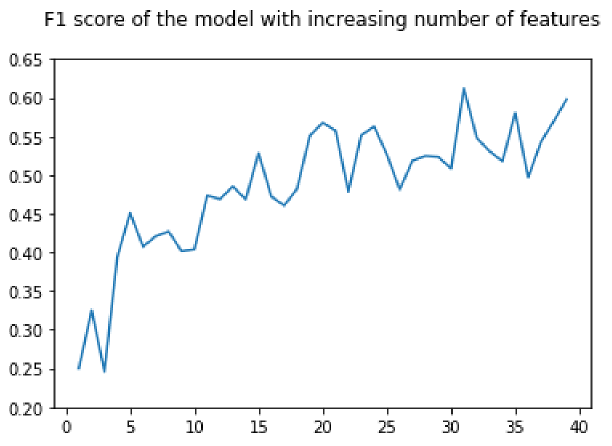
In [31]:

```
plt.plot(numvar, accuracy)
plt.yticks(np.arange(0.5, 0.9, step=0.05))
plt.title('Accuracy of the model with increasing number of features\n')
plt.show()
```



In [32]:

```
plt.plot(numvar, f1_score)
plt.yticks(np.arange(0.2, 0.7, step=0.05))
plt.title('F1 score of the model with increasing number of features\n')
plt.show()
```



It seems that 20 features are sufficient in order to have a good and stable predictive power on the test set. On the other hand, using all of the variables could lead to overfitting, the results on the test set are not stable and it could be a decrease both accuracy and F1 score on the validation set.

So I am going to use only the top 20 variables ranked by importance.
Here I am subsetting my dataset to the selected variables, splitting the Training and Testing set (80/20 ratio), scaling and trasforming each value and print the top values.

In [50]:

```
V = 20
for i in range(V):
    sel_features = [ranked_vars[i][1] for i in range(V)]
# New X feature matrix with V selected variables
X = d_dataset[sel_features]
# Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
# Feature Scaling. After scaling data are converted into arrays
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)

print(sel_features)
print(X.shape)
X.head()
```

```
['BLOOD SUGAR', 'BMI', 'EXPERIENCED_0', 'FAINT RATE', 'MONTH_11', 'MONTH_2', 'MONTH_3', 'MONTH_4',
'MONTH_7', 'MONTH_8', 'PHYSICAL  STATUS_0', 'PHYSICAL  STATUS_2', 'SHOES TYPE_0', 'SHOES TYPE_2',
'SHOES TYPE_3', 'SHOES TYPE_4', 'WATER INTAKE_0', 'WATER INTAKE_3', 'WEATHER CONDITION',
'MONTH_1']
(2845, 20)
```

Out[50]:

	BLOOD SUGAR	BMI	EXPERIENCED_0	FAINT RATE	MONTH_11	MONTH_2	MONTH_3	MONTH_4	MONTH_7	MONTH_8	PHYSICA STATUS_
0	5.13	17.990	0	0.01	0	0	0	0	0	1	0
1	6.15	20.704	1	0.01	1	0	0	0	0	0	0
2	6.36	19.156	1	0.01	0	0	0	0	0	0	0
3	6.62	16.802	0	0.02	0	0	0	0	1	0	0
5	5.96	20.042	1	0.01	0	0	0	0	0	0	1

Finally I am going to train the model and save it. (Random State = 0 in order to replicate the results)

In [51]:

```
# Train Predict Evaluate the final model and print results
logistic_class = LogisticRegression(random_state = 0)
lr_model = logistic_class.fit(X_train, y_train)
y_hat= lr_model.predict(X_test)

cm = confusion_matrix(y_test, y_hat)
tn = cm[0,0]
fn = cm[0,1]
fp = cm[1,0]
tp = cm[1,1]
precision = tp/(tp+fp)
recall = tp/(tp+fn)
f1 = 2*((precision*recall)/(precision+recall))
acc = np.sum(cm.diagonal()/np.sum(cm))
print('Logistic Regression Model with top 20 variables')
print('F1 =',f1,'\nAccuracy =',acc)
```

```
Logistic Regression Model with top 20 variables
F1 = 0.5396825396825397
Accuracy = 0.7451669595782074
```

The Logistic Regression Model with the 20 most significant variables has an accuracy of 75% and F1 score of 54% on this test set.

Importing the Validation Data

In order to apply our model I need to import the validation set and apply the same data-preprocessing pipeline to the new data.

In [52]:

```
# Import Validation Set 6th November

validation = pd.read_csv('Marathon_Eval_7984.csv')
print(validation.shape)
validation.head()
```

```
(7984, 15)
```

Out[52]:

	BMI	AGE	PHYSICAL STATUS	WEATHER CONDITION	FAINT RATE	SLOPE	SHOES TYPE	SEX	ELEVATION FACTOR	MONTH	WATER INTAKE	EXPERIENCED	BLOO SUGA
0	17.136	32	1	0.30	0.01	-2.84	4	0	294.5	4	1	0	5.82
1	17.476	30	0	0.16	0.01	-10.72	2	0	431.5	8	1	1	5.31
2	17.148	32	1	0.93	0.04	0.32	4	1	615.5	0	2	0	5.91

	BMI	AGE	PHYSICAL STATUS	WEATHER CONDITION	FAINT RATE	SLOPE	SHOES TYPE	SEX	ELEVATION FACTOR	MONTH	WATER INTAKE	EXPERIENCED	BLO SUG
2	17.148	32	1	0.93	0.04	0.32	4	1	615.5	0	2	0	5.91
3	17.966	26	1	0.37	0.02	3.32	3	1	569.5	7	1	0	6.51
4	18.044	22	1	0.16	0.01	-8.60	0	1	368.0	8	0	1	4.83

In [55]:

```
# Data preparation pipeline

features = (list(validation.columns.values))[0:15]

# Transforming missing data in HBR using 'median'
validation['HBR'].isnull().sum() # 2110 nan
validation['HBR'].fillna(validation['HBR'].median(), inplace=True)

categories = ['PHYSICAL  STATUS', 'SHOES TYPE', 'SEX', 'WATER INTAKE', 'EXPERIENCED', 'SLEEP CONDIT
ION', 'MONTH']

# Convert categorical variables into dummy variables
d_validation = pd.get_dummies(validation, columns = categories)

# New dataset dummies: 39 features
d_features = list(d_validation.columns.values)
print(d_features)
print('\n',d_validation.shape)
print('\nFeatures:v',len(d_features))
d_validation.head()
```

```
['BMI', 'AGE', 'WEATHER CONDITION', 'FAINT RATE', 'SLOPE', 'ELEVATION FACTOR', 'BLOOD SUGAR', 'HBR
', 'PHYSICAL  STATUS_0', 'PHYSICAL  STATUS_1', 'PHYSICAL  STATUS_2', 'SHOES TYPE_0', 'SHOES
TYPE_1', 'SHOES TYPE_2', 'SHOES TYPE_3', 'SHOES TYPE_4', 'SEX_0', 'SEX_1', 'WATER INTAKE_0', 'WATE
R INTAKE_1', 'WATER INTAKE_2', 'WATER INTAKE_3', 'EXPERIENCED_0', 'EXPERIENCED_1', 'SLEEP
CONDITION_0', 'SLEEP CONDITION_1', 'SLEEP CONDITION_2', 'MONTH_0', 'MONTH_1', 'MONTH_2',
'MONTH_3', 'MONTH_4', 'MONTH_5', 'MONTH_6', 'MONTH_7', 'MONTH_8', 'MONTH_9', 'MONTH_10',
'MONTH_11']

 (7984, 39)

Features: 39
```

Out[55]:

	BMI	AGE	WEATHER CONDITION	FAINT RATE	SLOPE	ELEVATION FACTOR	BLOOD SUGAR	HBR	PHYSICAL STATUS_0	PHYSICAL STATUS_1	...	MONTH_2	MONTH_3	M
0	17.136	32	0.30	0.01	-2.84	294.5	5.82	75.0	0	1	...	0	0	1
1	17.476	30	0.16	0.01	-10.72	431.5	5.31	91.0	1	0	...	0	0	0
2	17.148	32	0.93	0.04	0.32	615.5	5.91	75.0	0	1	...	0	0	0
3	17.966	26	0.37	0.02	3.32	569.5	6.51	75.0	0	1	...	0	0	0
4	18.044	22	0.16	0.01	-8.60	368.0	4.83	75.0	0	1	...	0	0	0

5 rows × 39 columns

Then I am going to subset the data to the selected number of features, scale and trasform the data.

In [56]:

```
# New X feature matrix with V selected variables
X_val = d_validation[sel_features]
# Feature Scaling. After scaling data are converted into arrays
sc_X = StandardScaler()
X_val = sc_X.fit_transform(X_val)
X_val.shape
```

Out[56]:

```
(7984, 20)
```

Finally I am going to predict the *'Failure'* outcome variable by applying the trained Logistic Regression model, attach the values to the validation dataset and export the .csv file for submission.

In [71]:

```
# Prediction
pred = lr_model.predict(X_val)
validation['Prediction'] = pred
validation.head()
```

Out[71]:

	BMI	AGE	PHYSICAL STATUS	WEATHER CONDITION	FAINT RATE	SLOPE	SHOES TYPE	SEX	ELEVATION FACTOR	MONTH	WATER INTAKE	EXPERIENCED	BLO SUG
0	17.136	32	1	0.30	0.01	-2.84	4	0	294.5	4	1	0	5.82
1	17.476	30	0	0.16	0.01	-10.72	2	0	431.5	8	1	1	5.31
2	17.148	32	1	0.93	0.04	0.32	4	1	615.5	0	2	0	5.91
3	17.966	26	1	0.37	0.02	3.32	3	1	569.5	7	1	0	6.51
4	18.044	22	1	0.16	0.01	-8.60	0	1	368.0	8	0	1	4.83

In [75]:

```
validation.to_csv('Kira_Kosareva_submission.csv',sep=',')
```