In [1]:
```
# dataframe management
import pandas as pd
# numerical computation
import numpy as np
# visualization library
import seaborn as sns
sns.set(style="white", color_codes=True)
sns.set_context(rc={"font.family":'sans',"font.size":24,"axes.titlesize":24,"axes.lab
elsize":24})
# import matplotlib and allow it to plot inline
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.linear_model import RidgeClassifier
#from sklearn.linear_model import Ridge,Lasso, LassoCV
from sklearn.model_selection import cross_val_score
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import scipy
from scipy.stats import skew
from sklearn.preprocessing import MinMaxScaler
import xgboost as xgb
from sklearn.decomposition import PCA
import tensorflow as tf
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import f1_score
```

In [2]:
```
import importlib #importlib.reload(WhatToReimport)
import hw5
importlib.reload(hw5)
```

Out[2]:
```
<module 'hw5' from 'C:\\Users\\andre\\Downloads\\cs412-hw\\hw5\\MachineLearningProje
ct\\hw5.py'>
```

In [3]:
```
d=hw5.Dataset()
```

Data exploration

In [4]:
```
d.data.describe()
```

Out[4]:

	Music	Slow songs or fast songs	Dance	Folk	Country	Classical music	Mus
count	1007.000000	1008.000000	1006.000000	1005.000000	1005.000000	1003.000000	1008.000
mean	4.731877	3.328373	3.113320	2.288557	2.123383	2.956132	2.761905
std	0.664049	0.833931	1.170568	1.138916	1.076136	1.252570	1.260845
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	5.000000	3.000000	2.000000	1.000000	1.000000	2.000000	2.000000
50%	5.000000	3.000000	3.000000	2.000000	2.000000	3.000000	3.000000
75%	5.000000	4.000000	4.000000	3.000000	3.000000	4.000000	4.000000
max	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000

8 rows × 139 columns

In [5]:
```
d.data.shape
```

Out[5]: (1010, 150)

Preprocessing

Missing values of the target feature

In [6]:
```
nulls = d.data.isnull().sum()
sorted([(x,y) for (x,y) in zip(nulls.index, nulls) if y>0], key=lambda x: x[1], rever
se=True)
```

```
Out[6]: [('Height', 20),
         ('Weight', 20),
         ('Passive sport', 15),
         ('Chemistry', 10),
         ('Geography', 9),
         ('Punk', 8),
         ('Latino', 8),
         ('Documentary', 8),
         ('Theatre', 8),
         ('Smoking', 8),
         ('Classical music', 7),
         ('Reggae, Ska', 7),
         ('Rock n roll', 7),
         ('Alternative', 7),
         ('Techno, Trance', 7),
         ('Countryside, outdoors', 7),
         ('Gardening', 7),
         ('Daily events', 7),
         ('Final judgement', 7),
         ('Criminal damage', 7),
         ('Compassion to animals', 7),
         ('Age', 7),
         ('Rock', 6),
         ('Swing, Jazz', 6),
         ('Movies', 6),
         ('PC', 6),
         ('Biology', 6),
         ('Reading', 6),
         ('Art exhibitions', 6),
         ('Writing', 6),
         ('Science and technology', 6),
         ('Friends versus money', 6),
         ('Giving', 6),
         ('Responding to a serious letter', 6),
         ('Number of siblings', 6),
         ('Gender', 6),
         ('Folk', 5),
         ('Country', 5),
         ('Psychology', 5),
         ('Economy Management', 5),
         ('Foreign languages', 5),
         ('Medicine', 5),
         ('Spiders', 5),
         ('Alcohol', 5),
         ('Prioritising workload', 5),
         ('Workaholism', 5),
         ('Self-criticism', 5),
         ('Empathy', 5),
         ('Socializing', 5),
         ('Energy levels', 5),
         ('Getting up', 5),
         ('Dance', 4),
         ('Hiphop, Rap', 4),
         ('Western', 4),
         ('Internet', 4),
         ('Cars', 4),
         ('Active sport', 4),
         ('Fun with friends', 4),
         ('Pets', 4),
         ('Reliability', 4),
         ('Loss of interest', 4),
         ('Funniness', 4),
         ('Decision making', 4),
         ('Judgment calls', 4),
         ('Hypochondria', 4),
```

```
('Cheating in school', 4),
('Mood swings', 4),
('Children', 4),
('Getting angry', 4),
('Happiness in life', 4),
('Small - big dogs', 4),
('Personality', 4),
('Finding lost valuables', 4),
('Questionnaires or polls', 4),
('Village - town', 4),
('House - block of flats', 4),
('Music', 3),
('Pop', 3),
('Metal or Hardrock', 3),
('Comedy', 3),
('Romantic', 3),
('Fantasy/Fairy tales', 3),
('Animated', 3),
('Mathematics', 3),
('Physics', 3),
('Religion', 3),
('Dancing', 3),
('Adrenaline sports', 3),
('Flying', 3),
('Heights', 3),
('Rats', 3),
('Healthy eating', 3),
('Writing notes', 3),
('Thinking ahead', 3),
('Elections', 3),
('Charity', 3),
('Waiting', 3),
('Appearence and gestures', 3),
('Unpopularity', 3),
('Life struggles', 3),
('Interests or hobbies', 3),
('Finances', 3),
('Entertainment spending', 3),
('Spending on looks', 3),
('Left - right handed', 3),
('Slow songs or fast songs', 2),
('Musical', 2),
('Horror', 2),
('Sci-fi', 2),
('War', 2),
('Action', 2),
('History', 2),
('Celebrities', 2),
('Shopping', 2),
('Darkness', 2),
('Borrowed stuff', 2),
('Changing the past', 2),
('God', 2),
('Punctuality', 2),
('Lying', 2),
('New environment', 2),
('Achievements', 2),
('Assertiveness', 2),
('Knowing the right people', 2),
('Public speaking', 2),
("Parents' advice", 2),
('Shopping centres', 2),
('Branded clothing', 2),
('Spending on healthy eating', 2),
('Only child', 2),
('Opera', 1),
```

```
 ('Thriller', 1),
 ('Politics', 1),
 ('Law', 1),
 ('Musical instruments', 1),
 ('Storm', 1),
 ('Ageing', 1),
 ('Dangerous dogs', 1),
 ('Fear of public speaking', 1),
 ('Keeping promises', 1),
 ('Fake', 1),
 ('Loneliness', 1),
 ('Health', 1),
 ('Education', 1)]
```

We have to manage all these missing values.
First of all I will remove all the rows that have the target feature "Empathy" to null because they have no use.

In [7]:
```
#removing the rows in which the Empathy attrivute is null
#they are not necessary for train or testing
nullsEmpathy = d.data["Empathy"].isnull().sum()
#nullsEmpathy = 5
print("Number of rows with Empathy that is null: "+str(nullsEmpathy))
d.data = d.data[d.data["Empathy"].notna()]
print("Number of rows with Empathy that is null after: "+str(d.data["Empathy"].isnull
().sum()))
```

```
Number of rows with Empathy that is null: 5
Number of rows with Empathy that is null after: 0
```

Dealing with the categorical variables

Now I have to deal with the categorical variables.
The first thing that I have to do is to impute the missing values of them. I will use the mode() (which is the most common value for each feature) to impute them.

In [8]:
```
categorical=d.data.select_dtypes(include="object", exclude="float")
```

In [9]:
```
d.data = d.data.select_dtypes(exclude="object")
```

In [10]:
```
categorical.mode().loc[0]
```

Out[10]:
```
Smoking                           tried smoking
Alcohol                           social drinker
Punctuality                  i am always on time
Lying                                 sometimes
Internet usage                 few hours a day
Gender                                   female
Left - right handed                right handed
Education                      secondary school
Only child                                   no
Village - town                             city
House - block of flats           block of flats
Name: 0, dtype: object
```

In [11]:
```
print(categorical.isnull().sum())
categorical = categorical.fillna(categorical.mode().loc[0])
print(categorical.isnull().sum())
```

```
Smoking                  8
Alcohol                  5
Punctuality              2
Lying                    2
Internet usage           0
Gender                   6
Left - right handed      3
Education                1
Only child               2
Village - town           4
House - block of flats   4
dtype: int64
Smoking                  0
Alcohol                  0
Punctuality              0
Lying                    0
Internet usage           0
Gender                   0
Left - right handed      0
Education                0
Only child               0
Village - town           0
House - block of flats   0
dtype: int64
```

From categorical to scale

From various attempts it turns out that one-hot encoding of all th variables leads to bad results.
From the theory we can understand this result because one hot encoding leads to have too many features and, moreover, the values of this categorical attributes are actually in a scale of values even if they are strings, to the best thing to do is to turn them in integers with a scale. (As done below)
I will do one-hot encoding only fot the binary features where the two values represents different things.

In [12]:
```
categorical.shape
```

Out[12]:
```
(1005, 11)
```

In [13]:
```
categorical.columns
```

Out[13]:
```
Index(['Smoking', 'Alcohol', 'Punctuality', 'Lying', 'Internet usage',
       'Gender', 'Left - right handed', 'Education', 'Only child',
       'Village - town', 'House - block of flats'],
      dtype='object')
```

In [14]:
```
categorical.columns=['Smoking', 'Alcohol', 'Punctuality', 'Lying', 'Internet_usage',
       'Gender', 'Left_right_handed', 'Education', 'Only_child',
       'Village_town', 'House_block_of_flats']
```

In [15]:
```
categorical.describe()
```

Out[15]:

	Smoking	Alcohol	Punctuality	Lying	Internet_usage	Gender	Left_right_handed	E
count	1005	1005	1005	1005	1005	1005	1005	1
unique	4	3	3	4	4	2	2	6
top	tried smoking	social drinker	i am always on time	sometimes	few hours a day	female	right handed	s s
freq	437	663	400	546	741	596	904	6

In [16]:
```
categorical.Smoking.unique()
```

Out[16]:
```
array(['never smoked', 'tried smoking', 'former smoker', 'current smoker'],
      dtype=object)
```

In [17]:
```
for row in categorical.itertuples():#range(len(categorical["Smoking"])):
    #print(row)
    #if(i==607 or i==722 or i==845 or i==858 or i==921 ):
    #    continue
    #print(row.Smoking)
    #print(row.Index)
    if(row.Smoking=="never smoked"):
        categorical['Smoking'][row.Index]=1
        continue
    if(row.Smoking=="tried smoking"):
        categorical['Smoking'][row.Index]=2
        continue
    if(row.Smoking=="former smoker"):
        categorical['Smoking'][row.Index]=3
        continue
    if(row.Smoking=="current smoker"):
        categorical['Smoking'][row.Index]=4
        continue
```

In [18]:
```
categorical.Smoking.unique()
```

Out[18]:
```
array([1, 2, 3, 4], dtype=object)
```

In [19]:
```
categorical.Alcohol.unique()
```

Out[19]:
```
array(['drink a lot', 'social drinker', 'never'], dtype=object)
```

In [20]:
```
for row in categorical.itertuples():
    if(row.Alcohol=="never"):
        categorical['Alcohol'][row.Index]=1
        continue
    if(row.Alcohol=="social drinker"):
        categorical['Alcohol'][row.Index]=2
        continue
    if(row.Alcohol=="drink a lot"):
        categorical['Alcohol'][row.Index]=3
        continue
```

In [21]:
```
categorical.Alcohol.unique()
```

Out[21]:
```
array([3, 2, 1], dtype=object)
```

In [22]:
```
categorical.Punctuality.unique()
```

Out[22]:
```
array(['i am always on time', 'i am often early',
       'i am often running late'], dtype=object)
```

In [23]:
```
for row in categorical.itertuples():
    if(row.Punctuality=="i am often running late"):
        categorical['Punctuality'][row.Index]=1
        continue
    if(row.Punctuality=="i am always on time"):
        categorical['Punctuality'][row.Index]=2
        continue
    if(row.Punctuality=="i am often early"):
        categorical['Punctuality'][row.Index]=3
        continue
```

In [24]:
```
categorical.Punctuality.unique()
```

Out[24]:
```
array([2, 3, 1], dtype=object)
```

In [25]:
```
categorical.Lying.unique()
```

Out[25]:
```
array(['never', 'sometimes', 'only to avoid hurting someone',
       'everytime it suits me'], dtype=object)
```

In [26]:
```
for row in categorical.itertuples():
    if(row.Lying=="everytime it suits me"):
        categorical['Lying'][row.Index]=1
        continue
    if(row.Lying=="sometimes"):
        categorical['Lying'][row.Index]=2
        continue
    if(row.Lying=="only to avoid hurting someone"):
        categorical['Lying'][row.Index]=3
        continue
    if(row.Lying=="never"):
        categorical['Lying'][row.Index]=4
        continue
```

In [27]:
```
categorical.Lying.unique()
```

Out[27]:
```
array([4, 2, 3, 1], dtype=object)
```

In [28]:
```
categorical.Internet_usage.unique()
```

Out[28]:
```
array(['few hours a day', 'most of the day', 'less than an hour a day',
       'no time at all'], dtype=object)
```

In [29]:
```
for row in categorical.itertuples():
    if(row.Internet_usage=="most of the day"):
        categorical['Internet_usage'][row.Index]=1
        continue
    if(row.Internet_usage=="few hours a day"):
        categorical['Internet_usage'][row.Index]=2
        continue
    if(row.Internet_usage=="less than an hour a day"):
        categorical['Internet_usage'][row.Index]=3
        continue
    if(row.Internet_usage=="no time at all"):
        categorical['Internet_usage'][row.Index]=4
        continue
```

In [30]:
```
categorical.Internet_usage.unique()
```

Out[30]:
```
array([2, 1, 3, 4], dtype=object)
```

In [31]:
```
categorical.Education.unique()
```

Out[31]:
```
array(['college/bachelor degree', 'secondary school', 'primary school',
       'masters degree', 'doctorate degree',
       'currently a primary school pupil'], dtype=object)
```

In [32]:
```
for row in categorical.itertuples():
    if(row.Education=="currently a primary school pupil"):
        categorical['Education'][row.Index]=1
        continue
    if(row.Education=="primary school"):
        categorical['Education'][row.Index]=2
        continue
    if(row.Education=="secondary school"):
        categorical['Education'][row.Index]=3
        continue
    if(row.Education=="college/bachelor degree"):
        categorical['Education'][row.Index]=4
        continue
    if(row.Education=="masters degree"):
        categorical['Education'][row.Index]=5
        continue
    if(row.Education=="doctorate degree"):
        categorical['Education'][row.Index]=6
        continue
```

In [33]:
```
categorical.Education.unique()
```

Out[33]:
```
array([4, 3, 2, 5, 6, 1], dtype=object)
```

In [34]:
```
categorical.describe()
```

Out[34]:

	Smoking	Alcohol	Punctuality	Lying	Internet_usage	Gender	Left_right_handed	Educa
count	1005	1005	1005	1005	1005	1005	1005	1005
unique	4	3	3	4	4	2	2	6
top	2	2	2	2	2	female	right handed	3
freq	437	663	400	546	741	596	904	619

One-hot encoding of categorical variables that are left

In [35]:
```
categorical.shape
```

Out[35]:
```
(1005, 11)
```

In [36]:
```
categorical["Smoking"]=categorical["Smoking"].astype("float64")
categorical["Alcohol"]=categorical["Alcohol"].astype("float64")
categorical["Punctuality"]=categorical["Punctuality"].astype("float64")
categorical["Lying"]=categorical["Lying"].astype("float64")
categorical["Internet_usage"]=categorical["Internet_usage"].astype("float64")
categorical["Education"]=categorical["Education"].astype("float64")
```

In [37]:
```
categorical.dtypes
```

Out[37]:
```
Smoking                  float64
Alcohol                  float64
Punctuality              float64
Lying                    float64
Internet_usage           float64
Gender                    object
Left_right_handed         object
Education                float64
Only_child                object
Village_town              object
House_block_of_flats      object
dtype: object
```

In [38]:
```
categorical2=categorical.select_dtypes(include="object", exclude="float64")
categorical = categorical.select_dtypes(exclude="object")
```

In [39]:
```
categoricalDummied = pd.get_dummies(categorical2)
```

In [40]:
```
categoricalDummied.shape
```

Out[40]:
```
(1005, 10)
```

Imputation of missing values for the numerical features

I wil use the mean value of each attribute to impute the value of missing values for numerical features

In [41]:
```
d.data=d.data.fillna(d.data.mean())
```

Outliers: Boxplot and Winsorizing

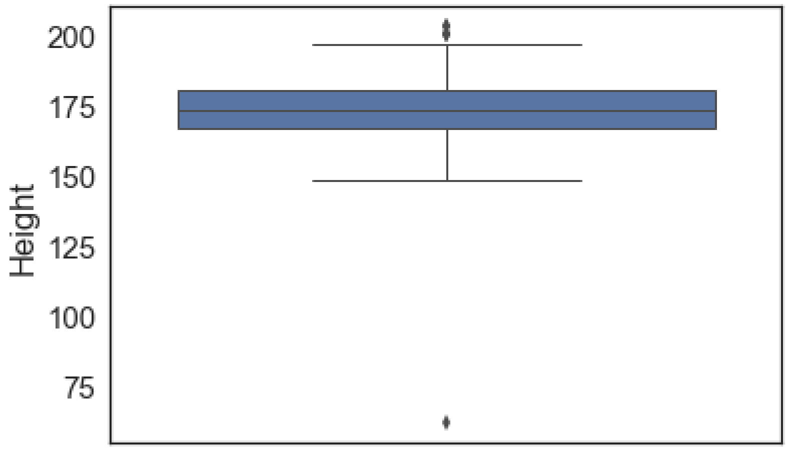
In [42]:
```
d.data.quantile(.99).sort_values(ascending=False).head(8)
```

Out[42]:
```
Height                 194.96
Weight                 102.92
Age                     29.00
Number of siblings       5.00
Geography                5.00
Religion                 5.00
Art exhibitions          5.00
Cars                     5.00
Name: 0.99, dtype: float64
```

In [43]:
```
def q(col, quant, f):
    t = d.data[col].quantile(quant)
    print(f'col {col} at {quant}-th quantile => {t}')
    d.data.loc[f(d.data[col], t), col] = t
```

```
In [44]: sns.set_context("notebook", font_scale=1.5, rc={"lines.linewidth": 1})
         sns.boxplot(y="Height", data=d.data)
```

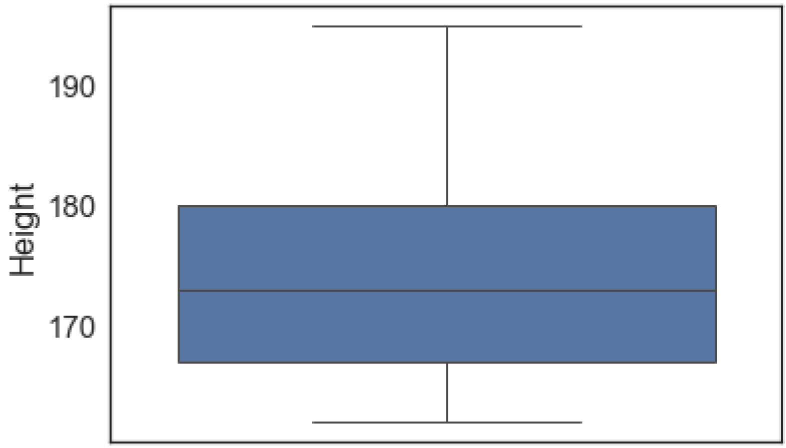
```
Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x247b34bd550>
```




```
In [45]: q("Height", .99, lambda x, y: x > y)
         q("Height", .1, lambda x,y: x < y)
         sns.boxplot(y="Height", data=d.data)
```

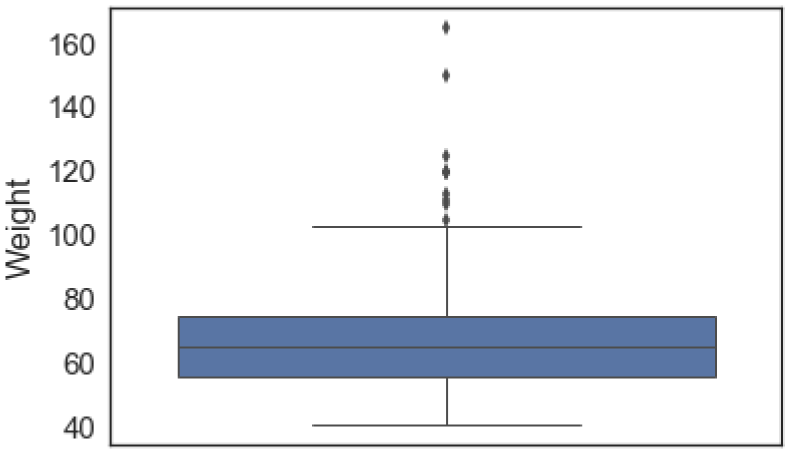
```
col Height at 0.99-th quantile => 194.96000000000004
col Height at 0.1-th quantile => 162.0
```

```
Out[45]: <matplotlib.axes._subplots.AxesSubplot at 0x247b34c10b8>
```




```
In [46]: sns.boxplot(y="Weight", data=d.data)
```

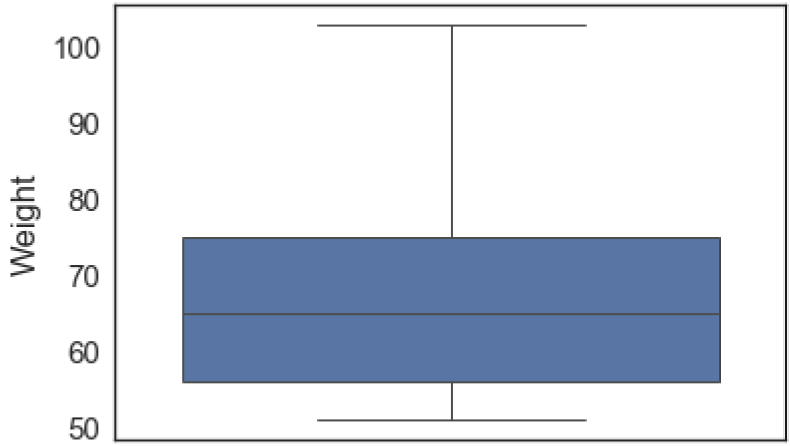
```
Out[46]: <matplotlib.axes._subplots.AxesSubplot at 0x247b3c099b0>
```



In [47]:
```python
q("Weight", .99, lambda x, y: x > y)
q("Weight", .1, lambda x,y: x < y)
sns.boxplot(y="Weight", data=d.data)
```

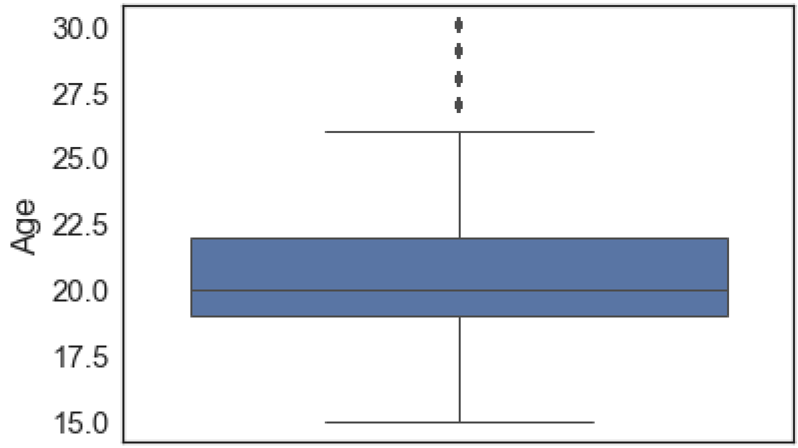
```
col Weight at 0.99-th quantile => 102.92000000000007
col Weight at 0.1-th quantile => 51.0
```

Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0x247b3c745c0>



In [48]:
```python
sns.boxplot(y="Age", data=d.data)
```

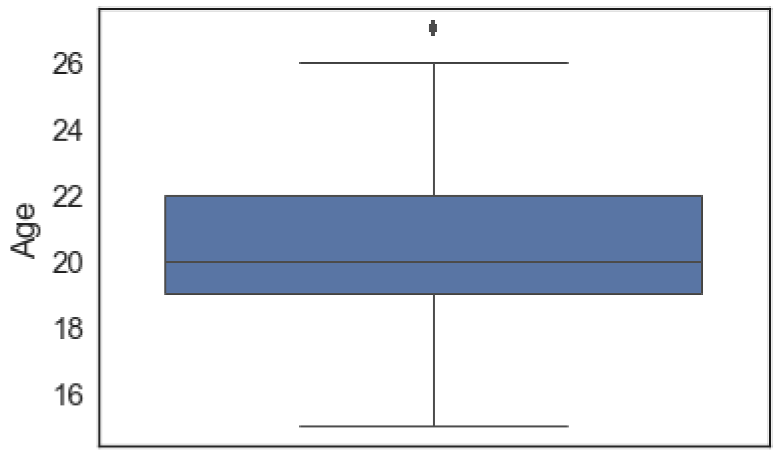
Out[48]: <matplotlib.axes._subplots.AxesSubplot at 0x247b3cc3748>



In [49]:
```python
q("Age", .95, lambda x, y: x > y)
#q("Age", .1, lambda x,y: x < y)
sns.boxplot(y="Age", data=d.data)
```

```
col Age at 0.95-th quantile => 27.0
```

Out[49]: <matplotlib.axes._subplots.AxesSubplot at 0x247b3d20080>



Normalization of Numerical Variables

In [50]:
```
scaler = MinMaxScaler(feature_range=(1, 5), copy=True)
scaled_df = scaler.fit_transform(d.data)
scaled_df = pd.DataFrame(scaled_df, columns=d.data.columns)
```

```
C:\Users\andre\Anaconda3\envs\cs412\lib\site-packages\sklearn\preprocessing\data.py:
323: DataConversionWarning: Data with input dtype int64, float64 were all converted
to float64 by MinMaxScaler.
  return self.partial_fit(X, y)
```

In [51]:
```
d.data=scaled_df
```

In [52]:
```
d.data= pd.concat([d.data,categoricalDummied,categorical],axis=1,join='inner')
```

In [53]:
```
d.data.describe()
```

Out[53]:

	Music	Slow songs or fast songs	Dance	Folk	Country	Classical music	Mus
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000
mean	4.737207	3.331662	3.112452	2.289450	2.124620	2.956691	2.764763
std	0.658470	0.831812	1.170802	1.137512	1.076897	1.246873	1.260678
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	5.000000	3.000000	2.000000	1.000000	1.000000	2.000000	2.000000
50%	5.000000	3.000000	3.000000	2.000000	2.000000	3.000000	3.000000
75%	5.000000	4.000000	4.000000	3.000000	3.000000	4.000000	4.000000
max	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000

8 rows × 155 columns

In [54]:
```
nulls = d.data.isnull().sum()
sorted([(x,y) for (x,y) in zip(nulls.index, nulls) if y>0], key=lambda x: x[1], rever
se=True)
```

Out[54]: []

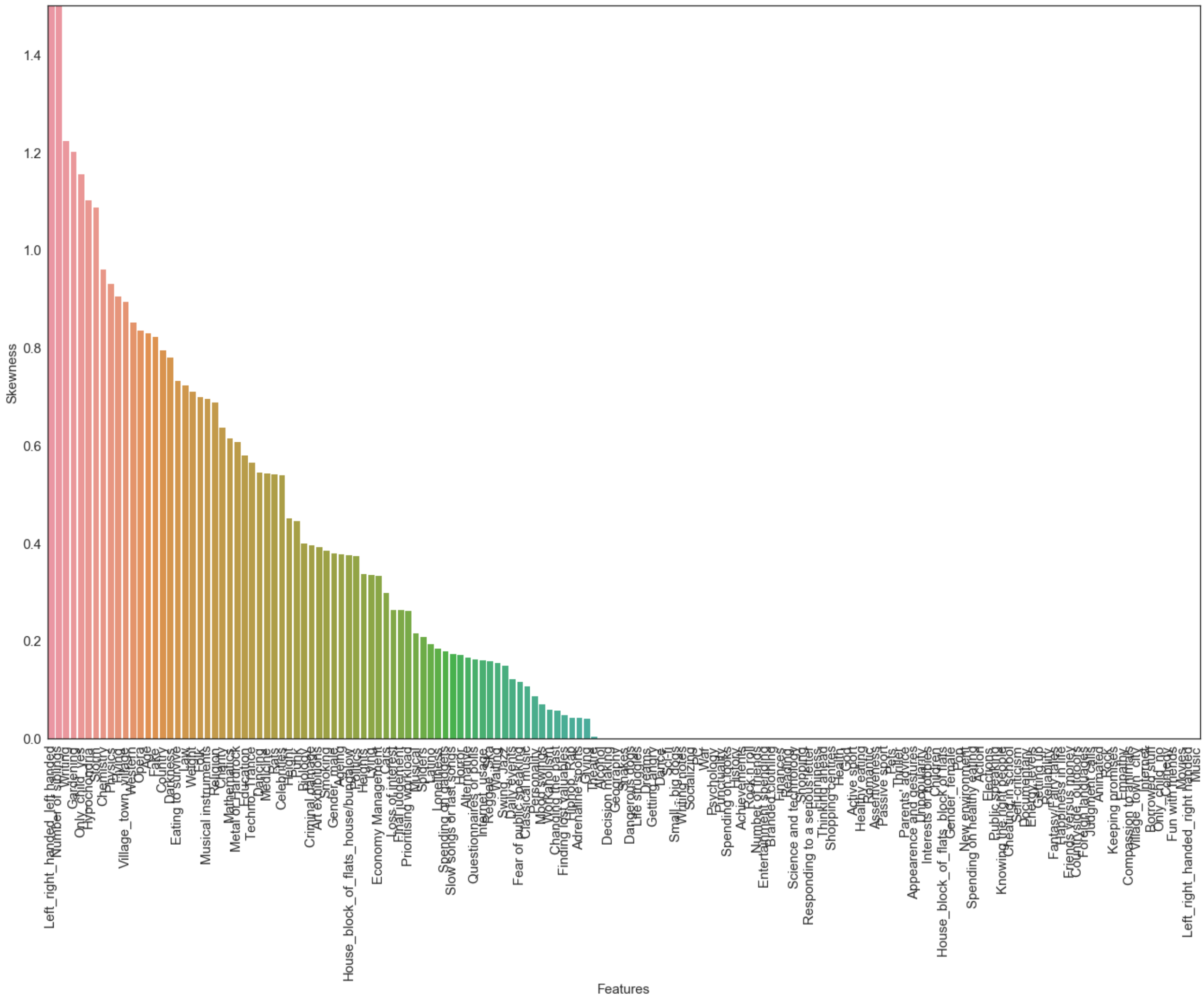
In [55]:
```
d.data.shape
```

Out[55]: (1000, 155)

In [56]:
```
# compute the skewness but only for non missing variables (we already imputed them bu
t just in case ...)
skewed_feats = d.data.apply(lambda x: skew(x.dropna()))
skewness = pd.DataFrame({"Variable":skewed_feats.index, "Skewness":skewed_feats.data
})
skewness = skewness.sort_values('Skewness', ascending=[0])
f, ax = plt.subplots(figsize=(23,15))
plt.xticks(rotation='90')
sns.barplot(x=skewness['Variable'], y=skewness['Skewness'])
plt.ylim(0,1.5)
plt.xlabel('Features', fontsize=15)
plt.ylabel('Skewness', fontsize=15)
plt.title('', fontsize=15)
```

```
C:\Users\andre\Anaconda3\envs\cs412\lib\site-packages\ipykernel_launcher.py:3: Futur
eWarning: Series.data is deprecated and will be removed in a future version
  This is separate from the ipykernel package so we can avoid doing imports until
```

Out[56]:
```
Text(0.5,1,'')
```



Training and Testing sets

In [57]:
```
X = d.data.drop(columns=['Empathy'])
Y = d.data['Empathy']
```

I have now to trasform the target feature from a scale from 1 to 5 to a binary variable 0 (if the vale is 1,2 or 3) and 1 (4 ot 5).

In [58]:
```
def getBinary(x):
    res=[]
    for i in range(len(x)):
        if(x[i]<=3):
            res.append(0)
        else:
            res.append(1)
    res = np.array(res)
    return res
```

Baseline

In [59]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, stratify=Y, test_size=0.2,
random_state=40)
```

I will use as baseline a dump predictor that predict always the most frequent.

In [60]:
```
Y_train=getBinary(Y_train.values)
Y_test=getBinary(Y_test.values)
Y=getBinary(Y.values)
```

In [61]:
```
def trainBaseline(x):
    return scipy.stats.mode(x)[0][0]
```

In [62]:
```
mode=trainBaseline(Y_train)
```

In [63]:
```
def predictBaseline(test,mostfrequent):
    res=[]
    for i in range(len(test)):
        res.append(mostfrequent)
    return np.array(res)
```

In [64]:
```
predictions=predictBaseline(X_test,mode)
```

In [65]:
```
np.mean(predictions==Y_test)
```

Out[65]:
```
0.665
```

Usign the 20% of my dataset as testing set, this base predictor has an accuracy of 66%

Model1

I start trying a Logistic Regression algoritm, I choose as solver the 'liblinear' one that should be one of the most suitable for binary classification in small databases.
I'll use the L2 norm for the penalization and a value of C very small. C is the inverse of regularization strength, like in support vector machines, smaller values specify stronger regularization.

In [66]:
```
from sklearn.linear_model import LogisticRegression
logReg = LogisticRegression(solver='liblinear',random_state=123, C=0.1,penalty='l2',
multi_class='ovr')
logReg.fit(X_train, Y_train)
```

Out[66]:
```
LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr',
          n_jobs=None, penalty='l2', random_state=123, solver='liblinear',
          tol=0.0001, verbose=0, warm_start=False)
```

In [67]:
```
predict_train = logReg.predict(X_train)
```

In [68]:
```
np.mean(predict_train == Y_train)
```

Out[68]:
```
0.8175
```

In [69]:
```
predict_test = logReg.predict(X_test)
```

In [70]:
```
np.mean(predict_test == Y_test)
```

Out[70]:
```
0.7
```

We have already achieved an accuracy of 82% on the training set and an accuracy of 70% on the testing one.

Now I will try to have a better idea of the real accuracy that that model can reach using a Stratified K-Fold cross validation.

In [71]:
```
X.shape
```

Out[71]:
```
(1000, 154)
```

In [72]:
```
np.mean(cross_val_score(logReg, X, Y, cv=150))
```

Out[72]:
```
0.6927777777777779
```

Model2

In [73]:
```
def svc_param_selection(X_t, Y_t, n):
    params = {'C': [0.001, 0.01, 0.1, 1, 10], 'gamma' : [0.001, 0.01, 0.1, 1]}
    gs = GridSearchCV(svm.SVC(kernel='rbf'), params, cv=n,n_jobs=-1)
    gs.fit(X_t, Y_t)
    return gs.best_params_
```

In [74]:
```
svc_param_selection(X_train,Y_train,20)
```

Out[74]:
```
{'C': 10, 'gamma': 0.01}
```

In [75]:
```
model2=svm.SVC(kernel='rbf',C= 10, gamma= 0.01)
model2.fit(X_train,Y_train)
```

Out[75]:
```
SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [76]:
```
prediction2=model2.predict(X_test)
```

In [77]:
```
np.mean(prediction2 == Y_test)
```
Out[77]:
```
0.72
```

Model 3

In [57]:
```
# Heavy to run:
rfc = RandomForestClassifier()
params = {'n_estimators': [4, 15,20,50,100,200,250,300,350],
        #'n_estimators': [4, 6, 9,15,20,50,100,150,200,250,300,350],
              #'max_features': ['log2', 'sqrt','auto'],
              'max_features': ['auto'],
              #'criterion': ['entropy', 'gini'],
              'criterion': ['gini'],
              'max_depth': [3, 5, 10,50,100,250],
              'min_samples_split': [2,3, 5,10,15],
              'min_samples_leaf': [1,5,8,18]
              #'max_depth': [2, 3, 5, 10,15,20,25,30,50,100],
              #'min_samples_split': [2, 3, 5,10],
              #'min_samples_leaf': [1,5,8]
             }
gs = GridSearchCV(clf, params,iid=False,cv=10,n_jobs=-1)
gs = gs.fit(X_train, Y_train)
rfc = gs.best_estimator_
rfc.fit(X_train, Y_train)
```
Out[57]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=100, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=5,
            min_weight_fraction_leaf=0.0, n_estimators=250, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [58]:
```
gs.best_params_
```
Out[58]:
```
{'criterion': 'gini',
 'max_depth': 100,
 'max_features': 'auto',
 'min_samples_leaf': 1,
 'min_samples_split': 5,
 'n_estimators': 250}
```

In [59]:
```
prediction3=rfc.predict(X_test)
```

In [60]:
```
np.mean(prediction3 == Y_test)
```
Out[60]:
```
0.7164179104477612
```

Redone with preprocessed data:

In [78]:
```
model3=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=100, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=5,
            min_weight_fraction_leaf=0.0, n_estimators=250, n_jobs=None,
            oob_score=False, random_state=129,
            warm_start=False)
model3.fit(X_train,Y_train)
predict3=model3.predict(X_test)
np.mean(predict3 == Y_test)
```

Out[78]: 0.76

In [79]:
```
f1_score(Y_test,predict3)
```

Out[79]: 0.8421052631578947

Model 4

In [80]:
```
xg_reg = xgb.XGBClassifier(n_estimators = 300)
```

In [81]:
```
xg_reg.fit(X_train,Y_train)

preds = xg_reg.predict(X_test)
```

In [82]:
```
xg_reg.score(X_test,Y_test)
```

Out[82]: 0.715

In [74]:
```
#heavy to run
xgboostClass = xgb.XGBClassifier()
parameters = {'n_estimators': [4, 15,20,50,100,200,250,300,350,500,1000],
              'objective': ['reg:logistic','binary:logistic'],
              'max_depth': [3, 5, 10,50,100,250],
              'learning_rate': [0.1,0.5,0.3,0.8,0.01,0.003],
              "subsample": [0.6, 0.4],
              "colsample_bytree": [0.7, 0.3],
              "gamma": [0, 0.5 ]
             }
```

In [75]:
```
grid_obj = GridSearchCV(xgboostClass, parameters,iid=False,cv=10,n_jobs=-1)
grid_obj = grid_obj.fit(X_train, Y_train)

# Set the clf to the best combination of parameters
clf = grid_obj.best_estimator_

# Fit the best algorithm to the data.
clf.fit(X_train, Y_train)
grid_obj.best_estimator_
```

Out[75]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=0.3, gamma=0.5, learning_rate=0.01,
       max_delta_step=0, max_depth=5, min_child_weight=1, missing=None,
       n_estimators=1000, n_jobs=1, nthread=None, objective='reg:logistic',
       random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
       seed=None, silent=True, subsample=0.6)
```

In [79]:
```
predict4=clf.predict(X_test)
np.mean(predict4 == Y_test) #
```

Out[79]: 0.7213930348258707

Redone with preprocessed data:

In [83]:
```
model4 = xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=0.3, gamma=0.5, learning_rate=0.001,
       max_delta_step=0, max_depth=5, min_child_weight=1, missing=None,
       n_estimators=6000, n_jobs=1, nthread=None, objective='reg:logistic',
       random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
       seed=None, silent=True, subsample=0.6)
```

In [84]:
```
model4.fit(X_train, Y_train)
```

Out[84]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=0.3, gamma=0.5, learning_rate=0.001,
       max_delta_step=0, max_depth=5, min_child_weight=1, missing=None,
       n_estimators=6000, n_jobs=1, nthread=None, objective='reg:logistic',
       random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
       seed=None, silent=True, subsample=0.6)
```

In [85]:
```
predict4=model4.predict(X_test)
```

In [86]:
```
np.mean(predict4 == Y_test)
```

Out[86]: 0.735

Model 5

In [87]:
```
model5 = RidgeClassifier(random_state=123)
model5 = model5.fit(X_train, Y_train)
predict5 = model5.predict(X_test)
np.mean(predict5==Y_test)
```

Out[87]: 0.705

Model 6

In [88]:
```
model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(6, activation=tf.nn.relu),
  tf.keras.layers.Dense(8, activation=tf.nn.tanh),
  tf.keras.layers.Dense(8, activation=tf.nn.tanh),
  tf.keras.layers.Dense(8, activation=tf.nn.tanh),
  tf.keras.layers.Dropout(0.3),
  tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(X_train.values, Y_train , epochs=200)
```

```
Epoch 1/200
800/800 [==============================] - 0s 533us/step - loss: 0.6494 - acc: 0.647
5
Epoch 2/200
800/800 [==============================] - 0s 20us/step - loss: 0.6401 - acc: 0.6687
Epoch 3/200
800/800 [==============================] - 0s 48us/step - loss: 0.6379 - acc: 0.6687
Epoch 4/200
800/800 [==============================] - 0s 31us/step - loss: 0.6343 - acc: 0.6537
Epoch 5/200
800/800 [==============================] - 0s 96us/step - loss: 0.6234 - acc: 0.6737
Epoch 6/200
800/800 [==============================] - 0s 54us/step - loss: 0.6329 - acc: 0.6550
Epoch 7/200
800/800 [==============================] - 0s 111us/step - loss: 0.6131 - acc: 0.676
3
Epoch 8/200
800/800 [==============================] - 0s 55us/step - loss: 0.6200 - acc: 0.6600
Epoch 9/200
800/800 [==============================] - 0s 56us/step - loss: 0.6121 - acc: 0.6913
Epoch 10/200
800/800 [==============================] - 0s 54us/step - loss: 0.5973 - acc: 0.6962
Epoch 11/200
800/800 [==============================] - 0s 66us/step - loss: 0.6108 - acc: 0.6775
Epoch 12/200
800/800 [==============================] - 0s 47us/step - loss: 0.5886 - acc: 0.6962
Epoch 13/200
800/800 [==============================] - 0s 55us/step - loss: 0.5956 - acc: 0.6913
Epoch 14/200
800/800 [==============================] - 0s 47us/step - loss: 0.5899 - acc: 0.7013
Epoch 15/200
800/800 [==============================] - 0s 34us/step - loss: 0.5894 - acc: 0.7025
Epoch 16/200
800/800 [==============================] - 0s 39us/step - loss: 0.5788 - acc: 0.6925
Epoch 17/200
800/800 [==============================] - 0s 59us/step - loss: 0.5745 - acc: 0.7050
Epoch 18/200
800/800 [==============================] - 0s 20us/step - loss: 0.5749 - acc: 0.7263
Epoch 19/200
800/800 [==============================] - 0s 39us/step - loss: 0.5615 - acc: 0.7162
Epoch 20/200
800/800 [==============================] - 0s 144us/step - loss: 0.5590 - acc: 0.737
5
Epoch 21/200
800/800 [==============================] - 0s 81us/step - loss: 0.5380 - acc: 0.7512
Epoch 22/200
800/800 [==============================] - 0s 107us/step - loss: 0.5274 - acc: 0.741
3
Epoch 23/200
800/800 [==============================] - 0s 59us/step - loss: 0.5352 - acc: 0.7488
Epoch 24/200
800/800 [==============================] - 0s 59us/step - loss: 0.5286 - acc: 0.7625
Epoch 25/200
800/800 [==============================] - 0s 39us/step - loss: 0.5351 - acc: 0.7425
Epoch 26/200
800/800 [==============================] - 0s 39us/step - loss: 0.5197 - acc: 0.7512
Epoch 27/200
800/800 [==============================] - 0s 59us/step - loss: 0.5025 - acc: 0.7762
Epoch 28/200
800/800 [==============================] - 0s 59us/step - loss: 0.5106 - acc: 0.7688
Epoch 29/200
800/800 [==============================] - 0s 39us/step - loss: 0.4925 - acc: 0.7787
Epoch 30/200
800/800 [==============================] - 0s 39us/step - loss: 0.4996 - acc: 0.7675
Epoch 31/200
```

```
800/800 [==============================] - 0s 39us/step - loss: 0.5012 - acc: 0.7675
Epoch 32/200
800/800 [==============================] - 0s 59us/step - loss: 0.4707 - acc: 0.8025
Epoch 33/200
800/800 [==============================] - 0s 20us/step - loss: 0.4480 - acc: 0.7987
Epoch 34/200
800/800 [==============================] - 0s 39us/step - loss: 0.4682 - acc: 0.7950
Epoch 35/200
800/800 [==============================] - 0s 39us/step - loss: 0.4599 - acc: 0.7987
Epoch 36/200
800/800 [==============================] - 0s 78us/step - loss: 0.4575 - acc: 0.8100
Epoch 37/200
800/800 [==============================] - 0s 78us/step - loss: 0.4235 - acc: 0.8362
Epoch 38/200
800/800 [==============================] - 0s 59us/step - loss: 0.4392 - acc: 0.8137
Epoch 39/200
800/800 [==============================] - 0s 39us/step - loss: 0.4313 - acc: 0.8300
Epoch 40/200
800/800 [==============================] - 0s 39us/step - loss: 0.4124 - acc: 0.8337
Epoch 41/200
800/800 [==============================] - 0s 39us/step - loss: 0.4162 - acc: 0.8350
Epoch 42/200
800/800 [==============================] - 0s 20us/step - loss: 0.4096 - acc: 0.8250
Epoch 43/200
800/800 [==============================] - 0s 39us/step - loss: 0.4051 - acc: 0.8337
Epoch 44/200
800/800 [==============================] - 0s 20us/step - loss: 0.3908 - acc: 0.8463
Epoch 45/200
800/800 [==============================] - 0s 39us/step - loss: 0.4067 - acc: 0.8325
Epoch 46/200
800/800 [==============================] - 0s 39us/step - loss: 0.3875 - acc: 0.8488
Epoch 47/200
800/800 [==============================] - 0s 47us/step - loss: 0.3869 - acc: 0.8450
Epoch 48/200
800/800 [==============================] - 0s 59us/step - loss: 0.3634 - acc: 0.8500
Epoch 49/200
800/800 [==============================] - 0s 39us/step - loss: 0.3582 - acc: 0.8687
Epoch 50/200
800/800 [==============================] - 0s 39us/step - loss: 0.3677 - acc: 0.8538
Epoch 51/200
800/800 [==============================] - 0s 39us/step - loss: 0.3424 - acc: 0.8788
Epoch 52/200
800/800 [==============================] - 0s 39us/step - loss: 0.3339 - acc: 0.8800
Epoch 53/200
800/800 [==============================] - 0s 39us/step - loss: 0.3306 - acc: 0.8675
Epoch 54/200
800/800 [==============================] - 0s 39us/step - loss: 0.3533 - acc: 0.8638
Epoch 55/200
800/800 [==============================] - 0s 39us/step - loss: 0.3242 - acc: 0.8875
Epoch 56/200
800/800 [==============================] - 0s 20us/step - loss: 0.3245 - acc: 0.8888
Epoch 57/200
800/800 [==============================] - 0s 39us/step - loss: 0.3259 - acc: 0.8837
Epoch 58/200
800/800 [==============================] - 0s 39us/step - loss: 0.3458 - acc: 0.8775
Epoch 59/200
800/800 [==============================] - 0s 20us/step - loss: 0.3345 - acc: 0.8837
Epoch 60/200
800/800 [==============================] - 0s 20us/step - loss: 0.3105 - acc: 0.9000
Epoch 61/200
800/800 [==============================] - 0s 39us/step - loss: 0.3360 - acc: 0.8900
Epoch 62/200
800/800 [==============================] - 0s 20us/step - loss: 0.3227 - acc: 0.8738
Epoch 63/200
800/800 [==============================] - 0s 39us/step - loss: 0.3080 - acc: 0.9012
Epoch 64/200
```

```
800/800 [==============================] - 0s 20us/step - loss: 0.2815 - acc: 0.9088
Epoch 65/200
800/800 [==============================] - 0s 39us/step - loss: 0.2741 - acc: 0.9163
Epoch 66/200
800/800 [==============================] - 0s 39us/step - loss: 0.2721 - acc: 0.9113
Epoch 67/200
800/800 [==============================] - 0s 20us/step - loss: 0.2636 - acc: 0.9150
Epoch 68/200
800/800 [==============================] - 0s 39us/step - loss: 0.2632 - acc: 0.9213
Epoch 69/200
800/800 [==============================] - 0s 39us/step - loss: 0.2647 - acc: 0.9138
Epoch 70/200
800/800 [==============================] - 0s 20us/step - loss: 0.2578 - acc: 0.9312
Epoch 71/200
800/800 [==============================] - 0s 39us/step - loss: 0.2531 - acc: 0.9300
Epoch 72/200
800/800 [==============================] - 0s 20us/step - loss: 0.2477 - acc: 0.9275
Epoch 73/200
800/800 [==============================] - 0s 39us/step - loss: 0.2596 - acc: 0.9200
Epoch 74/200
800/800 [==============================] - 0s 20us/step - loss: 0.2866 - acc: 0.9025
Epoch 75/200
800/800 [==============================] - 0s 39us/step - loss: 0.2698 - acc: 0.9138
Epoch 76/200
800/800 [==============================] - 0s 20us/step - loss: 0.2605 - acc: 0.9125
Epoch 77/200
800/800 [==============================] - 0s 39us/step - loss: 0.2473 - acc: 0.9275
Epoch 78/200
800/800 [==============================] - 0s 20us/step - loss: 0.2303 - acc: 0.9325
Epoch 79/200
800/800 [==============================] - 0s 20us/step - loss: 0.2765 - acc: 0.9050
Epoch 80/200
800/800 [==============================] - 0s 39us/step - loss: 0.2440 - acc: 0.9237
Epoch 81/200
800/800 [==============================] - 0s 20us/step - loss: 0.2396 - acc: 0.9300
Epoch 82/200
800/800 [==============================] - 0s 39us/step - loss: 0.2226 - acc: 0.9375
Epoch 83/200
800/800 [==============================] - 0s 19us/step - loss: 0.2237 - acc: 0.9462
Epoch 84/200
800/800 [==============================] - 0s 20us/step - loss: 0.2369 - acc: 0.9375
Epoch 85/200
800/800 [==============================] - 0s 39us/step - loss: 0.2497 - acc: 0.9300
Epoch 86/200
800/800 [==============================] - 0s 20us/step - loss: 0.2270 - acc: 0.9375
Epoch 87/200
800/800 [==============================] - 0s 39us/step - loss: 0.2357 - acc: 0.9150
Epoch 88/200
800/800 [==============================] - 0s 20us/step - loss: 0.2261 - acc: 0.9337
Epoch 89/200
800/800 [==============================] - 0s 39us/step - loss: 0.1934 - acc: 0.9487
Epoch 90/200
800/800 [==============================] - 0s 20us/step - loss: 0.2019 - acc: 0.9462
Epoch 91/200
800/800 [==============================] - 0s 20us/step - loss: 0.2081 - acc: 0.9425
Epoch 92/200
800/800 [==============================] - 0s 42us/step - loss: 0.1951 - acc: 0.9487
Epoch 93/200
800/800 [==============================] - 0s 27us/step - loss: 0.1991 - acc: 0.9513
Epoch 94/200
800/800 [==============================] - 0s 20us/step - loss: 0.1933 - acc: 0.9475
Epoch 95/200
800/800 [==============================] - 0s 20us/step - loss: 0.1990 - acc: 0.9450
Epoch 96/200
800/800 [==============================] - 0s 39us/step - loss: 0.1977 - acc: 0.9462
Epoch 97/200
```

```
800/800 [==============================] - 0s 20us/step - loss: 0.2154 - acc: 0.9400
Epoch 98/200
800/800 [==============================] - 0s 20us/step - loss: 0.2131 - acc: 0.9225
Epoch 99/200
800/800 [==============================] - 0s 20us/step - loss: 0.1781 - acc: 0.9513
Epoch 100/200
800/800 [==============================] - 0s 20us/step - loss: 0.1761 - acc: 0.9575
Epoch 101/200
800/800 [==============================] - 0s 39us/step - loss: 0.1778 - acc: 0.9612
Epoch 102/200
800/800 [==============================] - 0s 20us/step - loss: 0.1822 - acc: 0.9550
Epoch 103/200
800/800 [==============================] - 0s 39us/step - loss: 0.1739 - acc: 0.9563
Epoch 104/200
800/800 [==============================] - 0s 20us/step - loss: 0.1841 - acc: 0.9563
Epoch 105/200
800/800 [==============================] - 0s 20us/step - loss: 0.1878 - acc: 0.9462
Epoch 106/200
800/800 [==============================] - 0s 39us/step - loss: 0.2129 - acc: 0.9200
Epoch 107/200
800/800 [==============================] - 0s 20us/step - loss: 0.1781 - acc: 0.9487
Epoch 108/200
800/800 [==============================] - 0s 39us/step - loss: 0.1720 - acc: 0.9550
Epoch 109/200
800/800 [==============================] - 0s 20us/step - loss: 0.1980 - acc: 0.9462
Epoch 110/200
800/800 [==============================] - 0s 20us/step - loss: 0.1674 - acc: 0.9575
Epoch 111/200
800/800 [==============================] - 0s 39us/step - loss: 0.1448 - acc: 0.9688
Epoch 112/200
800/800 [==============================] - 0s 20us/step - loss: 0.1434 - acc: 0.9662
Epoch 113/200
800/800 [==============================] - 0s 39us/step - loss: 0.1546 - acc: 0.9600
Epoch 114/200
800/800 [==============================] - 0s 20us/step - loss: 0.1672 - acc: 0.9600
Epoch 115/200
800/800 [==============================] - 0s 39us/step - loss: 0.1786 - acc: 0.9525
Epoch 116/200
800/800 [==============================] - 0s 20us/step - loss: 0.1712 - acc: 0.9563
Epoch 117/200
800/800 [==============================] - 0s 20us/step - loss: 0.1895 - acc: 0.9462
Epoch 118/200
800/800 [==============================] - 0s 39us/step - loss: 0.1574 - acc: 0.9650
Epoch 119/200
800/800 [==============================] - 0s 20us/step - loss: 0.1695 - acc: 0.9587
Epoch 120/200
800/800 [==============================] - 0s 39us/step - loss: 0.1998 - acc: 0.9413
Epoch 121/200
800/800 [==============================] - 0s 20us/step - loss: 0.1601 - acc: 0.9612
Epoch 122/200
800/800 [==============================] - 0s 20us/step - loss: 0.1754 - acc: 0.9575
Epoch 123/200
800/800 [==============================] - 0s 20us/step - loss: 0.1368 - acc: 0.9675
Epoch 124/200
800/800 [==============================] - 0s 20us/step - loss: 0.1431 - acc: 0.9650
Epoch 125/200
800/800 [==============================] - 0s 39us/step - loss: 0.1381 - acc: 0.9725
Epoch 126/200
800/800 [==============================] - 0s 20us/step - loss: 0.1352 - acc: 0.9763
Epoch 127/200
800/800 [==============================] - 0s 39us/step - loss: 0.1424 - acc: 0.9688
Epoch 128/200
800/800 [==============================] - 0s 20us/step - loss: 0.1485 - acc: 0.9713
Epoch 129/200
800/800 [==============================] - 0s 20us/step - loss: 0.1284 - acc: 0.9750
Epoch 130/200
```

```
800/800 [==============================] - 0s 39us/step - loss: 0.1271 - acc: 0.9788
Epoch 131/200
800/800 [==============================] - 0s 20us/step - loss: 0.1328 - acc: 0.9725
Epoch 132/200
800/800 [==============================] - 0s 39us/step - loss: 0.1161 - acc: 0.9688
Epoch 133/200
800/800 [==============================] - 0s 20us/step - loss: 0.1358 - acc: 0.9675
Epoch 134/200
800/800 [==============================] - 0s 39us/step - loss: 0.1210 - acc: 0.9775
Epoch 135/200
800/800 [==============================] - 0s 20us/step - loss: 0.1203 - acc: 0.9775
Epoch 136/200
800/800 [==============================] - 0s 20us/step - loss: 0.1167 - acc: 0.9763
Epoch 137/200
800/800 [==============================] - 0s 25us/step - loss: 0.1254 - acc: 0.9788
Epoch 138/200
800/800 [==============================] - 0s 39us/step - loss: 0.1340 - acc: 0.9725
Epoch 139/200
800/800 [==============================] - 0s 20us/step - loss: 0.1205 - acc: 0.9788
Epoch 140/200
800/800 [==============================] - 0s 39us/step - loss: 0.1276 - acc: 0.9775
Epoch 141/200
800/800 [==============================] - 0s 20us/step - loss: 0.1154 - acc: 0.9800
Epoch 142/200
800/800 [==============================] - 0s 20us/step - loss: 0.1108 - acc: 0.9800
Epoch 143/200
800/800 [==============================] - 0s 39us/step - loss: 0.1155 - acc: 0.9775
Epoch 144/200
800/800 [==============================] - 0s 20us/step - loss: 0.1412 - acc: 0.9637
Epoch 145/200
800/800 [==============================] - 0s 39us/step - loss: 0.1891 - acc: 0.9400
Epoch 146/200
800/800 [==============================] - 0s 20us/step - loss: 0.3654 - acc: 0.8950
Epoch 147/200
800/800 [==============================] - 0s 20us/step - loss: 0.1940 - acc: 0.9438
Epoch 148/200
800/800 [==============================] - 0s 39us/step - loss: 0.1970 - acc: 0.9375
Epoch 149/200
800/800 [==============================] - 0s 20us/step - loss: 0.2302 - acc: 0.9200
Epoch 150/200
800/800 [==============================] - 0s 39us/step - loss: 0.2584 - acc: 0.9288
Epoch 151/200
800/800 [==============================] - 0s 20us/step - loss: 0.1699 - acc: 0.9600
Epoch 152/200
800/800 [==============================] - 0s 20us/step - loss: 0.2280 - acc: 0.9363
Epoch 153/200
800/800 [==============================] - 0s 39us/step - loss: 0.1809 - acc: 0.9550
Epoch 154/200
800/800 [==============================] - 0s 20us/step - loss: 0.1341 - acc: 0.9738
Epoch 155/200
800/800 [==============================] - 0s 39us/step - loss: 0.1270 - acc: 0.9725
Epoch 156/200
800/800 [==============================] - 0s 20us/step - loss: 0.1821 - acc: 0.9537
Epoch 157/200
800/800 [==============================] - 0s 20us/step - loss: 0.2387 - acc: 0.9325
Epoch 158/200
800/800 [==============================] - 0s 39us/step - loss: 0.2074 - acc: 0.9387
Epoch 159/200
800/800 [==============================] - 0s 20us/step - loss: 0.1742 - acc: 0.9500
Epoch 160/200
800/800 [==============================] - 0s 39us/step - loss: 0.1739 - acc: 0.9625
Epoch 161/200
800/800 [==============================] - 0s 20us/step - loss: 0.1771 - acc: 0.9550
Epoch 162/200
800/800 [==============================] - 0s 20us/step - loss: 0.1705 - acc: 0.9612
Epoch 163/200
```

```
800/800 [==============================] - 0s 39us/step - loss: 0.1659 - acc: 0.9637
Epoch 164/200
800/800 [==============================] - 0s 20us/step - loss: 0.1511 - acc: 0.9662
Epoch 165/200
800/800 [==============================] - 0s 39us/step - loss: 0.1654 - acc: 0.9675
Epoch 166/200
800/800 [==============================] - 0s 20us/step - loss: 0.1653 - acc: 0.9612
Epoch 167/200
800/800 [==============================] - 0s 20us/step - loss: 0.1669 - acc: 0.9513
Epoch 168/200
800/800 [==============================] - 0s 39us/step - loss: 0.1833 - acc: 0.9500
Epoch 169/200
800/800 [==============================] - 0s 20us/step - loss: 0.1597 - acc: 0.9637
Epoch 170/200
800/800 [==============================] - 0s 39us/step - loss: 0.1446 - acc: 0.9713
Epoch 171/200
800/800 [==============================] - 0s 20us/step - loss: 0.1465 - acc: 0.9688
Epoch 172/200
800/800 [==============================] - 0s 20us/step - loss: 0.1520 - acc: 0.9675
Epoch 173/200
800/800 [==============================] - 0s 39us/step - loss: 0.1527 - acc: 0.9713
Epoch 174/200
800/800 [==============================] - 0s 20us/step - loss: 0.1552 - acc: 0.9700
Epoch 175/200
800/800 [==============================] - 0s 39us/step - loss: 0.2141 - acc: 0.9387
Epoch 176/200
800/800 [==============================] - 0s 20us/step - loss: 0.1549 - acc: 0.9688
Epoch 177/200
800/800 [==============================] - 0s 39us/step - loss: 0.1350 - acc: 0.9738
Epoch 178/200
800/800 [==============================] - 0s 20us/step - loss: 0.1342 - acc: 0.9775
Epoch 179/200
800/800 [==============================] - 0s 62us/step - loss: 0.1317 - acc: 0.9738
Epoch 180/200
800/800 [==============================] - 0s 47us/step - loss: 0.1333 - acc: 0.9750
Epoch 181/200
800/800 [==============================] - 0s 41us/step - loss: 0.1313 - acc: 0.9713
Epoch 182/200
800/800 [==============================] - 0s 37us/step - loss: 0.1462 - acc: 0.9637
Epoch 183/200
800/800 [==============================] - 0s 37us/step - loss: 0.1497 - acc: 0.9625
Epoch 184/200
800/800 [==============================] - 0s 37us/step - loss: 0.1308 - acc: 0.9738
Epoch 185/200
800/800 [==============================] - 0s 45us/step - loss: 0.1345 - acc: 0.9713
Epoch 186/200
800/800 [==============================] - 0s 40us/step - loss: 0.1156 - acc: 0.9763
Epoch 187/200
800/800 [==============================] - 0s 55us/step - loss: 0.1253 - acc: 0.9788
Epoch 188/200
800/800 [==============================] - 0s 39us/step - loss: 0.1205 - acc: 0.9763
Epoch 189/200
800/800 [==============================] - 0s 39us/step - loss: 0.1254 - acc: 0.9812
Epoch 190/200
800/800 [==============================] - 0s 20us/step - loss: 0.1117 - acc: 0.9800
Epoch 191/200
800/800 [==============================] - 0s 39us/step - loss: 0.1246 - acc: 0.9800
Epoch 192/200
800/800 [==============================] - 0s 20us/step - loss: 0.1138 - acc: 0.9800
Epoch 193/200
800/800 [==============================] - 0s 39us/step - loss: 0.1161 - acc: 0.9788
Epoch 194/200
800/800 [==============================] - 0s 39us/step - loss: 0.1165 - acc: 0.9800
Epoch 195/200
800/800 [==============================] - 0s 20us/step - loss: 0.1183 - acc: 0.9812
Epoch 196/200
```

```
800/800 [==============================] - 0s 39us/step - loss: 0.1191 - acc: 0.9812
Epoch 197/200
800/800 [==============================] - 0s 39us/step - loss: 0.1013 - acc: 0.9812
Epoch 198/200
800/800 [==============================] - 0s 20us/step - loss: 0.1055 - acc: 0.9812
Epoch 199/200
800/800 [==============================] - 0s 39us/step - loss: 0.1141 - acc: 0.9812
Epoch 200/200
800/800 [==============================] - 0s 20us/step - loss: 0.1140 - acc: 0.9812
```

Out[88]:
```
<tensorflow.python.keras.callbacks.History at 0x247b3845ef0>
```

In [89]:
```
predictionNN=model.predict(X_test.values)
res=[]
for i in range(len(predictionNN)):
    if(predictionNN[i]<=0.5):
        res.append(0)
    else:
        res.append(1)
predictionNN = np.array(res)
np.mean(predictionNN==Y_test)
```

Out[89]:
```
0.59
```

The bad behavior of the Neural Network is expected and it is because of the small dimension of the dataset.

Model 7

In [90]:
```
model7=AdaBoostClassifier(model3,n_estimators=2,random_state=123)
model7.fit(X_train,Y_train)
```

Out[90]:
```
AdaBoostClassifier(algorithm='SAMME.R',
          base_estimator=RandomForestClassifier(bootstrap=True, class_weight=None, c
riterion='gini',
            max_depth=100, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=5,
            min_weight_fraction_leaf=0.0, n_estimators=250, n_jobs=None,
            oob_score=False, random_state=129, verbose=0, warm_start=False),
          learning_rate=1.0, n_estimators=2, random_state=123)
```

In [91]:
```
pred7=model7.predict(X_test)
np.mean(pred7==Y_test)
```

Out[91]:
```
0.725
```

In [92]:
```
np.mean(cross_val_score(model7, X_test, Y_test, cv=15))
```

Out[92]:
```
0.7211843711843712
```

In [93]:
```
f1_score(Y_test,pred7)
```

Out[93]:
```
0.8220064724919094
```

Dimensionality reduction

PCA

In [94]:
```
X.shape
```
Out[94]: (1000, 154)

In [95]:
```
X_train.shape
```
Out[95]: (800, 154)

In [96]:
```
pca = PCA(n_components=130)
X_trainPCA = pca.fit_transform(X_train)
X_testPCA = pca.fit_transform(X_test)
print(X_trainPCA.shape)
```
```
(800, 130)
```

In [97]:
```
model=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=100, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=5,
            min_weight_fraction_leaf=0.0, n_estimators=250, n_jobs=None,
            oob_score=False, random_state=223,
            warm_start=False)
model.fit(X_trainPCA,Y_train)
predict=model.predict(X_testPCA)
np.mean(predict == Y_test)
```
Out[97]: 0.665

Tuning of PCA

In [98]:
```
for i in [50,80,100,115,130,150]:
    pca = PCA(n_components=i)
    X_trainPCA = pca.fit_transform(X_train)
    X_testPCA = pca.fit_transform(X_test)
    print("number of feature of PCA: "+str(X_trainPCA.shape[1]))
    model=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=100, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=5,
            min_weight_fraction_leaf=0.0, n_estimators=250, n_jobs=None,
            oob_score=False, random_state=223,
            warm_start=False)
    model.fit(X_trainPCA,Y_train)
    predict6=model.predict(X_testPCA)
    print(np.mean(predict6 == Y_test))
    print("\n")
```

```
number of feature of PCA: 50
0.655


number of feature of PCA: 80
0.675


number of feature of PCA: 100
0.67


number of feature of PCA: 115
0.67


number of feature of PCA: 130
0.665


number of feature of PCA: 150
0.665
```

Feature Selection

In [99]:
```
model_simple = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='g
ini',
            max_depth=100, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=5,
            min_weight_fraction_leaf=0.0, n_estimators=250, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
model_simple = model_simple.fit(X_train, Y_train)
```

In [100]:

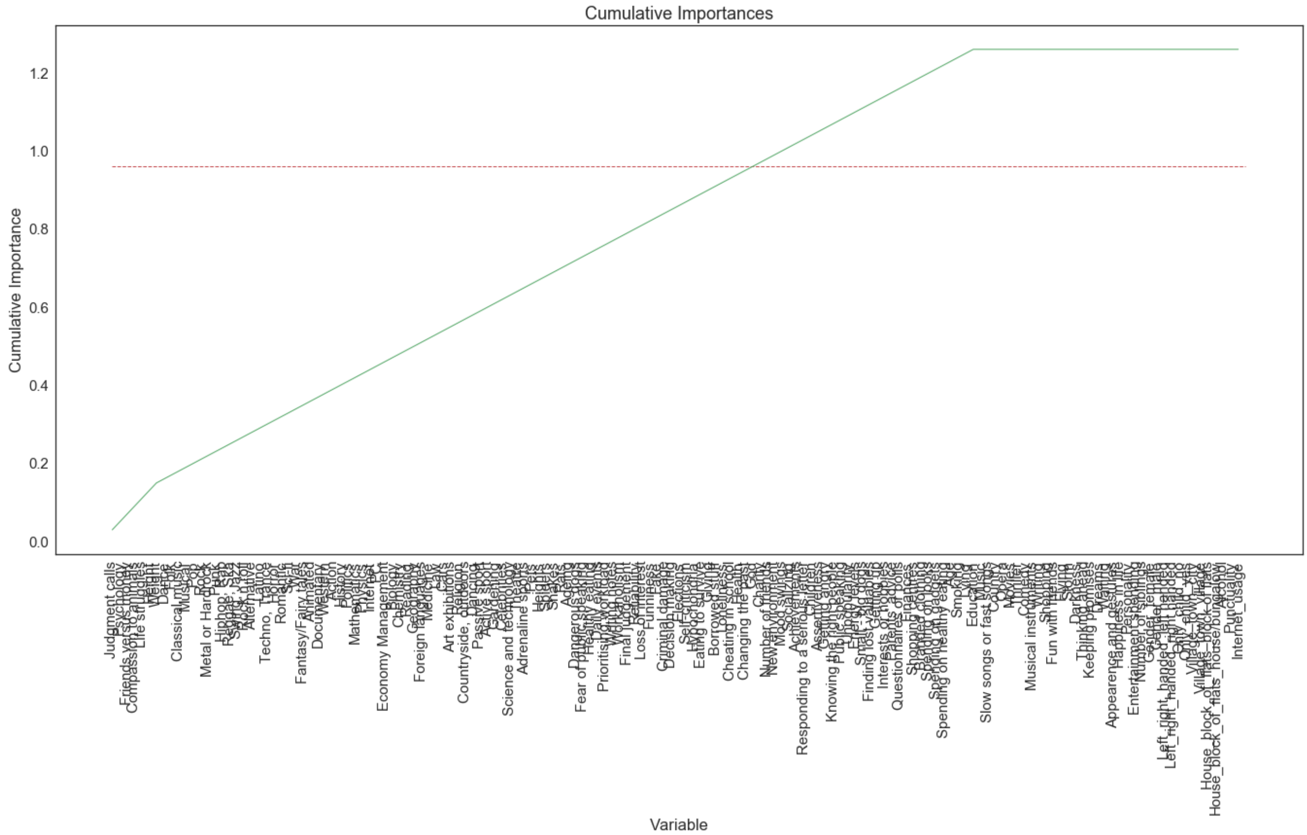
```
# Get numerical feature importances
importances = list(model_simple.feature_importances_)

# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 2)) for feature, importance in zip
(X_train, importances)]

# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = Tru
e)

# Print out the feature and importances
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_importance
s];
```

```
Variable: Judgment calls       Importance: 0.03
Variable: Psychology           Importance: 0.02
Variable: Friends versus money Importance: 0.02
Variable: Compassion to animals Importance: 0.02
Variable: Life struggles       Importance: 0.02
Variable: Height               Importance: 0.02
Variable: Weight               Importance: 0.02
Variable: Dance                Importance: 0.01
Variable: Folk                 Importance: 0.01
Variable: Classical music      Importance: 0.01
Variable: Musical              Importance: 0.01
Variable: Pop                  Importance: 0.01
Variable: Rock                 Importance: 0.01
Variable: Metal or Hardrock    Importance: 0.01
Variable: Punk                 Importance: 0.01
Variable: Hiphop, Rap          Importance: 0.01
Variable: Reggae, Ska          Importance: 0.01
Variable: Swing, Jazz          Importance: 0.01
Variable: Rock n roll          Importance: 0.01
Variable: Alternative          Importance: 0.01
Variable: Latino               Importance: 0.01
Variable: Techno, Trance       Importance: 0.01
Variable: Horror               Importance: 0.01
Variable: Romantic             Importance: 0.01
Variable: Sci-fi               Importance: 0.01
Variable: War                  Importance: 0.01
Variable: Fantasy/Fairy tales  Importance: 0.01
Variable: Animated             Importance: 0.01
Variable: Documentary          Importance: 0.01
Variable: Western              Importance: 0.01
Variable: Action               Importance: 0.01
Variable: History              Importance: 0.01
Variable: Politics             Importance: 0.01
Variable: Mathematics          Importance: 0.01
Variable: Physics              Importance: 0.01
Variable: Internet             Importance: 0.01
Variable: PC                   Importance: 0.01
Variable: Economy Management   Importance: 0.01
Variable: Biology              Importance: 0.01
Variable: Chemistry            Importance: 0.01
Variable: Reading              Importance: 0.01
Variable: Geography            Importance: 0.01
Variable: Foreign languages    Importance: 0.01
Variable: Medicine             Importance: 0.01
Variable: Law                  Importance: 0.01
Variable: Cars                 Importance: 0.01
Variable: Art exhibitions      Importance: 0.01
Variable: Religion             Importance: 0.01
Variable: Countryside, outdoors Importance: 0.01
Variable: Dancing              Importance: 0.01
Variable: Passive sport        Importance: 0.01
Variable: Active sport         Importance: 0.01
Variable: Gardening            Importance: 0.01
Variable: Celebrities          Importance: 0.01
Variable: Science and technology Importance: 0.01
Variable: Theatre              Importance: 0.01
Variable: Adrenaline sports    Importance: 0.01
Variable: Pets                 Importance: 0.01
Variable: Heights              Importance: 0.01
Variable: Spiders              Importance: 0.01
Variable: Snakes               Importance: 0.01
Variable: Rats                 Importance: 0.01
Variable: Ageing               Importance: 0.01
Variable: Dangerous dogs       Importance: 0.01
Variable: Fear of public speaking Importance: 0.01
```

```
Variable: Healthy eating       Importance: 0.01
Variable: Daily events         Importance: 0.01
Variable: Prioritising workload Importance: 0.01
Variable: Writing notes        Importance: 0.01
Variable: Workaholism          Importance: 0.01
Variable: Final judgement      Importance: 0.01
Variable: Reliability          Importance: 0.01
Variable: Loss of interest     Importance: 0.01
Variable: Funniness            Importance: 0.01
Variable: Fake                 Importance: 0.01
Variable: Criminal damage      Importance: 0.01
Variable: Decision making      Importance: 0.01
Variable: Elections            Importance: 0.01
Variable: Self-criticism       Importance: 0.01
Variable: Hypochondria         Importance: 0.01
Variable: Eating to survive    Importance: 0.01
Variable: Giving               Importance: 0.01
Variable: Borrowed stuff       Importance: 0.01
Variable: Loneliness           Importance: 0.01
Variable: Cheating in school   Importance: 0.01
Variable: Health               Importance: 0.01
Variable: Changing the past    Importance: 0.01
Variable: God                  Importance: 0.01
Variable: Charity              Importance: 0.01
Variable: Number of friends    Importance: 0.01
Variable: New environment      Importance: 0.01
Variable: Mood swings          Importance: 0.01
Variable: Socializing          Importance: 0.01
Variable: Achievements         Importance: 0.01
Variable: Responding to a serious letter Importance: 0.01
Variable: Children             Importance: 0.01
Variable: Assertiveness        Importance: 0.01
Variable: Getting angry        Importance: 0.01
Variable: Knowing the right people Importance: 0.01
Variable: Public speaking      Importance: 0.01
Variable: Unpopularity         Importance: 0.01
Variable: Energy levels        Importance: 0.01
Variable: Small - big dogs     Importance: 0.01
Variable: Finding lost valuables Importance: 0.01
Variable: Getting up           Importance: 0.01
Variable: Interests or hobbies Importance: 0.01
Variable: Parents' advice      Importance: 0.01
Variable: Questionnaires or polls Importance: 0.01
Variable: Finances             Importance: 0.01
Variable: Shopping centres     Importance: 0.01
Variable: Branded clothing     Importance: 0.01
Variable: Spending on looks    Importance: 0.01
Variable: Spending on gadgets  Importance: 0.01
Variable: Spending on healthy eating Importance: 0.01
Variable: Age                  Importance: 0.01
Variable: Smoking              Importance: 0.01
Variable: Lying                Importance: 0.01
Variable: Education            Importance: 0.01
Variable: Music                Importance: 0.0
Variable: Slow songs or fast songs Importance: 0.0
Variable: Country              Importance: 0.0
Variable: Opera                Importance: 0.0
Variable: Movies               Importance: 0.0
Variable: Thriller             Importance: 0.0
Variable: Comedy               Importance: 0.0
Variable: Musical instruments  Importance: 0.0
Variable: Writing              Importance: 0.0
Variable: Shopping             Importance: 0.0
Variable: Fun with friends     Importance: 0.0
Variable: Flying               Importance: 0.0
Variable: Storm                Importance: 0.0
```

```
Variable: Darkness             Importance: 0.0
Variable: Thinking ahead       Importance: 0.0
Variable: Keeping promises     Importance: 0.0
Variable: Dreams               Importance: 0.0
Variable: Waiting              Importance: 0.0
Variable: Appearence and gestures Importance: 0.0
Variable: Happiness in life    Importance: 0.0
Variable: Personality          Importance: 0.0
Variable: Entertainment spending Importance: 0.0
Variable: Number of siblings   Importance: 0.0
Variable: Gender_female        Importance: 0.0
Variable: Gender_male          Importance: 0.0
Variable: Left_right_handed_left handed Importance: 0.0
Variable: Left_right_handed_right handed Importance: 0.0
Variable: Only_child_no        Importance: 0.0
Variable: Only_child_yes       Importance: 0.0
Variable: Village_town_city    Importance: 0.0
Variable: Village_town_village Importance: 0.0
Variable: House_block_of_flats_block of flats Importance: 0.0
Variable: House_block_of_flats_house/bungalow Importance: 0.0
Variable: Alcohol              Importance: 0.0
Variable: Punctuality          Importance: 0.0
Variable: Internet_usage       Importance: 0.0
```

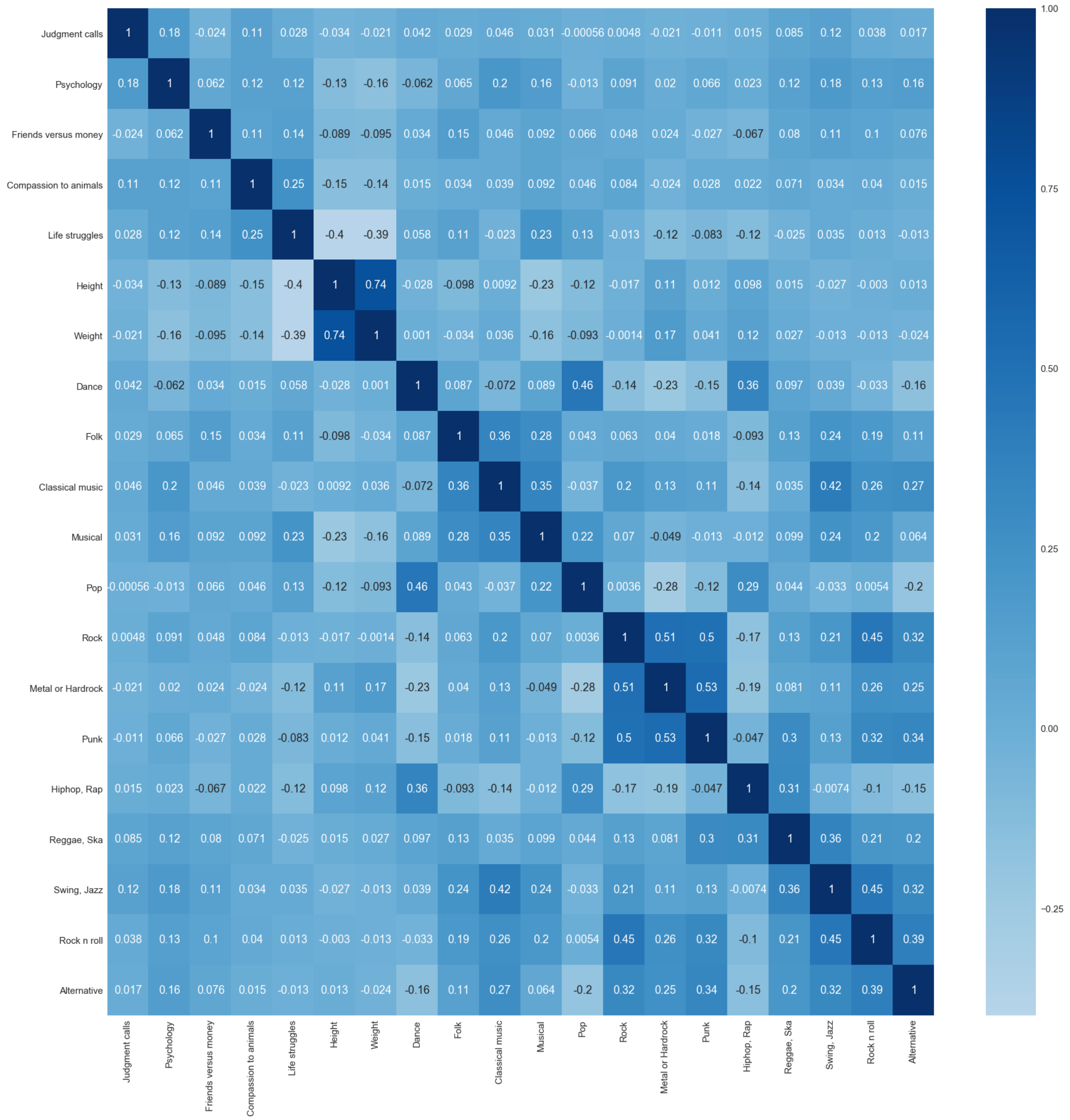
In [101]:
```
# List of x locations for plotting
x_values = list(range(len(importances)))

# List of features sorted from most to least important
sorted_importances = [importance[1] for importance in feature_importances]
sorted_features = [importance[0] for importance in feature_importances]

# Cumulative importances
cumulative_importances = np.cumsum(sorted_importances)

fig = plt.figure(figsize = (23,10))
# Make a line graph
plt.plot(x_values, cumulative_importances, 'g-')

# Draw line at 96% of importance retained
plt.hlines(y = 0.96, xmin=0, xmax=len(sorted_importances), color = 'r', linestyles =
'dashed')

# Format x ticks and labels
plt.xticks(x_values, sorted_features, rotation = 'vertical')

# Axis labels and title
plt.xlabel('Variable'); plt.ylabel('Cumulative Importance'); plt.title('Cumulative Im
portances');
```



In [102]:
```
f=[]
for i in range(120):
    f.append(feature_importances[i][0])
f=np.array(f)
```

In [103]:

```python
cov=X_train[f[:20]].corr(method='pearson')
#cm = sns.clustermap(cov, annot=True, center=0, cmap="Blues", figsize=(100, 100))
#cm.cax.set_visible(False)
fig, ax = plt.subplots(figsize=(30,30))
cm=sns.heatmap(cov, annot=True, center=0, cmap="Blues")
```



In [104]:

```python
X_train[f].shape
```

Out[104]: (800, 120)

In [105]:

```python
model3_featureSelection=RandomForestClassifier(bootstrap=True, class_weight=None, cri
terion='gini',
            max_depth=100, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=5,
            min_weight_fraction_leaf=0.0, n_estimators=250, n_jobs=None,
            oob_score=False, random_state=223,
            warm_start=False)
model3_featureSelection.fit(X_train[f],Y_train)
predict3_featureSelection=model3_featureSelection.predict(X_test[f])
np.mean(predict3_featureSelection == Y_test)
```

Out[105]: 0.725

In [106]:
```
xgb_featureSelection = xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_
bylevel=1,
       colsample_bytree=0.3, gamma=0.5, learning_rate=0.001,
       max_delta_step=0, max_depth=5, min_child_weight=1, missing=None,
       n_estimators=6000, n_jobs=-1, nthread=None, objective='reg:logistic',
       random_state=123, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
       seed=None, silent=True, subsample=0.6)
xgb_featureSelection.fit(X_train[f], Y_train)
pred=xgb_featureSelection.predict(X_test[f])
np.mean(pred == Y_test)
```

Out[106]: 0.72