In [1]:
```
import pandas as pd
from sklearn.metrics import mean_squared_error
from sklearn import linear_model

data = pd.read_csv("2008_births.csv")
data
```

Out[1]:

| | INST | RPLACE | RCOUNTY | PLURAL | BDATE | BMONTH | BDAY | BYEAR | SEX | RACE | ... | MOTHE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 6800 | 68 | 1 | 2008-01-01 | 1 | 1 | 2008 | 2 | 1 | ... | |
| **1** | 1 | 160 | 1 | 1 | 2008-01-02 | 1 | 2 | 2008 | 2 | 2 | ... | |
| **2** | 1 | 190 | 1 | 1 | 2008-01-02 | 1 | 2 | 2008 | 1 | 1 | ... | |
| **3** | 1 | 4100 | 41 | 1 | 2008-01-03 | 1 | 3 | 2008 | 2 | 1 | ... | |
| **4** | 1 | 160 | 1 | 1 | 2008-01-03 | 1 | 3 | 2008 | 2 | 1 | ... | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **133417** | 1 | 2000 | 20 | 1 | 2008-12-19 | 12 | 19 | 2008 | 1 | 1 | ... | |
| **133418** | 1 | 2000 | 20 | 1 | 2008-12-22 | 12 | 22 | 2008 | 2 | 1 | ... | |
| **133419** | 1 | 2600 | 26 | 1 | 2008-12-26 | 12 | 26 | 2008 | 1 | 1 | ... | |
| **133420** | 1 | 2000 | 20 | 1 | 2008-12-30 | 12 | 30 | 2008 | 2 | 1 | ... | |
| **133421** | 1 | 8100 | 81 | 1 | 2008-12-31 | 12 | 31 | 2008 | 1 | 0 | ... | |

133422 rows × 125 columns

In [2]:
```
import numpy as np
from sklearn.model_selection import train_test_split
```

In [3]:
```
#birthweight is what we want to predict - change this to single target
birth_weight = data[['BPOUND', 'BOUNCE']]
```

In [4]:
```
#PCA might be a good technique to select predictors

#note that PCA performs best when data is normalized (range b/w 0 and 1)

#It is possible to use categorical and continuous predictors
#for a regression problem. My understanding is you need to make
#dummy variables for the binary predictors.

#Variables that we will need to deal with:
# BDATE, HISPMOM, HISPDAD
```

In [5]:
```
#Attempting PCA on data
#for now I drop the BDATE, HISPMOM AND HISPDAD
data_drop = data.drop(["BDATE", "HISPMOM", "HISPDAD", "BOUNCE", "BPOUND"], axis = 1)
```

In [6]:
```
#get a list of columns in pandas object
names_of_data = data_drop.columns.tolist()

#shuffle = false prevents data split being different everytime
X_train, X_test, y_train, y_test = train_test_split(data_drop, birth_weight, test_size

#split test into validate and test, again making sure the data is always the same for
#X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.25, shuf

#Normalizing the data
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

#running the actual PCA
from sklearn.decomposition import PCA

pca = PCA()
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

#relief f algorithm - sorting features
```

In [7]:
```
explained_variance = pca.explained_variance_ratio_
print(len(explained_variance))
print(explained_variance)
```

```
120
[3.89856404e-02 3.41089470e-02 3.05310536e-02 2.86690661e-02
 2.39828710e-02 2.07124228e-02 1.81565355e-02 1.70798239e-02
 1.68763289e-02 1.59961073e-02 1.56220702e-02 1.35585642e-02
 1.30251204e-02 1.13087563e-02 1.10457470e-02 1.09427186e-02
 1.06522571e-02 1.03091115e-02 1.02034145e-02 1.01603763e-02
 9.98586972e-03 9.86404012e-03 9.78778404e-03 9.61165628e-03
 9.46902421e-03 9.40630729e-03 9.25092446e-03 9.20797437e-03
 9.16882901e-03 9.11728971e-03 9.08471022e-03 9.05880935e-03
 8.91090184e-03 8.85632587e-03 8.83902308e-03 8.82366452e-03
 8.73228213e-03 8.70972804e-03 8.64475483e-03 8.63888132e-03
 8.60300393e-03 8.57402898e-03 8.54542908e-03 8.51710741e-03
 8.50298288e-03 8.46111398e-03 8.42122923e-03 8.39505222e-03
 8.37548109e-03 8.34046815e-03 8.29732609e-03 8.28947627e-03
 8.24895028e-03 8.22904830e-03 8.20238682e-03 8.12690154e-03
 8.11341630e-03 8.08291392e-03 8.07851589e-03 8.03763212e-03
 8.01473052e-03 7.96613523e-03 7.90999598e-03 7.89944166e-03
 7.83600377e-03 7.82191448e-03 7.78512254e-03 7.75691445e-03
 7.69956508e-03 7.66449230e-03 7.60968558e-03 7.58920895e-03
 7.51965207e-03 7.50072699e-03 7.41546041e-03 7.36057792e-03
 7.17224177e-03 7.11636014e-03 7.01568819e-03 6.99311496e-03
 6.88970752e-03 6.80287045e-03 6.71667348e-03 6.59331242e-03
 6.56011619e-03 6.39097514e-03 6.21038587e-03 6.13263995e-03
 6.02101475e-03 5.88755078e-03 5.62716616e-03 5.49427350e-03
 5.42691648e-03 5.30849077e-03 5.16759622e-03 4.77164460e-03
 4.64430993e-03 4.52817477e-03 4.35785968e-03 4.10170975e-03
 3.95902522e-03 3.70091254e-03 3.10259706e-03 2.89941030e-03
 2.48400831e-03 1.79876640e-03 1.72329185e-03 9.56485073e-04
 9.01290670e-04 6.15880756e-04 5.54770674e-04 2.94096489e-04
 1.90827613e-04 3.83559595e-08 7.68856188e-33 7.62231993e-33
 2.16597016e-33 1.43573650e-33 2.23811207e-34 1.98989560e-34]
```

In [8]:
```
#Explained variance prints the variance each principal component contributes.
#As we can see, the last 5 contribute very little (maybe we can get rid of?)

#We also want to check for linearity between the input predictors and the output
#If there is high colinearity, then we want to use ridge regression - A variant of li

#Correlation indicates strength and direction of a linear relationship. let's use thi
```

In [9]:
```
# change y tro be combine ounces and pounds
y_train = y_train['BPOUND'] + y_train['BOUNCE']*0.0625
y_test = y_test['BPOUND'] + y_test['BOUNCE']*0.0625
```

## From now on is Bayesian

In [10]:
```
#Train
Las = linear_model.Lasso(alpha=10,max_iter=100000)
Las.fit(X_train, y_train)
#Try a prediction
LassoPredictedValues = Las.predict(X_test)
```

In [11]:
```
#show the mean square for comparison
mean_squared_error(y_test, LassoPredictedValues)
```

Out[11]: 5.508931548874832

In [ ]:
```
```