Group 5: Nishal Dave, Mollie Li, Wei Dai, Ajiboye Olaoye

In [121]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('seaborn-white')

from sklearn.model_selection import KFold

from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Lasso
from sklearn.linear_model import LassoCV
from sklearn.metrics import accuracy_score

from itertools import combinations
from itertools import combinations_with_replacement

!pip install glmnet
import glmnet as gln

import pydot
from IPython.display import Image
from sklearn.externals.six import StringIO
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier, export_graphviz
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier, BaggingRegressor, RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error,confusion_matrix, classification_report
```

```
Requirement already satisfied: glmnet in /usr/local/lib/python3.7/dist-packages (2.2.1)
Requirement already satisfied: numpy>=1.9.2 in /usr/local/lib/python3.7/dist-packages (from glmnet) (1.19.5)
Requirement already satisfied: scipy>=0.14.1 in /usr/local/lib/python3.7/dist-packages (from glmnet) (1.4.1)
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.7/dist-packages (from glmnet) (1.0.1)
Requirement already satisfied: scikit-learn>=0.18.0 in /usr/local/lib/python3.7/dist-packages (from glmnet) (0.22.2.post1)
```

$Question\ (1)$

(a)

In [122]:
```
df = pd.read_csv("Credit.csv")
df
```

Out[122]:

	ID	Income	Limit	Rating	Cards	Age	Education	Gender	Student	Married	Ethnicity	Balance
0	1	14.891	3606	283	2	34	11	Male	No	Yes	Caucasian	333
1	2	106.025	6645	483	3	82	15	Female	Yes	Yes	Asian	903
2	3	104.593	7075	514	4	71	11	Male	No	No	Asian	580
3	4	148.924	9504	681	3	36	11	Female	No	No	Asian	964
4	5	55.882	4897	357	2	68	16	Male	No	Yes	Caucasian	331
...	...	...	...	...	...	...	...	...	...	...	...	...
395	396	12.096	4100	307	3	32	13	Male	No	Yes	Caucasian	560
396	397	13.364	3838	296	5	65	17	Male	No	No	African American	480
397	398	57.872	4171	321	5	67	12	Female	No	Yes	Caucasian	138
398	399	37.728	2525	192	1	44	13	Male	No	Yes	Caucasian	0
399	400	18.701	5524	415	5	64	7	Female	No	No	Asian	966

400 rows × 12 columns

In [123]:
```
df.isna().any()
#Yields no missing observations
```

Out[123]:
```
ID            False
Income        False
Limit         False
Rating        False
Cards         False
Age           False
Education     False
Gender        False
Student       False
Married       False
Ethnicity     False
Balance       False
dtype: bool
```

In [124]:
```python
#Create dummies for:
  #cards, student, married, ethnicity

df.drop("ID",axis=1,inplace=True)
df.dropna()

#Dummy for Cards
#df.loc[:,"Cards:2"] = [1 if x==2 else 0 for x in df["Cards"]]
#df.loc[:,"Cards:3"] = [1 if x==3 else 0 for x in df["Cards"]]
#df.loc[:,"Cards:4"] = [1 if x==4 else 0 for x in df["Cards"]]
#df.loc[:,"Cards:5"] = [1 if x==5 else 0 for x in df["Cards"]]
#df.loc[:,"Cards:6"] = [1 if x==6 else 0 for x in df["Cards"]]
#df.loc[:,"Cards:7"] = [1 if x==7 else 0 for x in df["Cards"]]
#df.loc[:,"Cards:8"] = [1 if x==8 else 0 for x in df["Cards"]]
#df.loc[:,"Cards:9"] = [1 if x==9 else 0 for x in df["Cards"]]
#df.drop(columns="Cards",inplace=True)

#Dummy for Student
df.loc[:,"Student"] = [1 if x=="Yes" else 0 for x in df["Student"]]

#Dummy for Married
df.loc[:,"Married"] = [1 if x=="Yes" else 0 for x in df["Married"]]

#Dummy for Ethnicity
df.loc[:,"Asian"] = [1 if x=="Asian" else 0 for x in df["Ethnicity"]]
df.loc[:,"African American"] = [1 if x=="African American" else 0 for x in df["Ethnicity"]]
df.drop(columns="Ethnicity",inplace=True)

#Dummy for Gender
df.loc[:,"Male"] = [1 if x=="Male" else 0 for x in df["Gender"]]
df.drop(columns="Gender",inplace=True)

VartoStdze = ["Income","Limit","Rating","Age","Education","Cards"]
scale = StandardScaler()
for i in VartoStdze:
    df[i]=scale.fit_transform(df[i].values.reshape(-1,1))
```

```
In [125]: df
```

Out[125]:

	Income	Limit	Rating	Cards	Age	Education	Student	Married	Balance	Asian	African American	Male
0	-0.861583	-0.489999	-0.465539	-0.699130	-1.257674	-0.784930	0	1	333	0	0	0
1	1.727437	0.828261	0.828703	0.031032	1.528451	0.496588	1	1	903	1	0	0
2	1.686756	1.014787	1.029311	0.761194	0.889964	-0.784930	0	0	580	1	0	0
3	2.946152	2.068440	2.110003	0.031032	-1.141586	-0.784930	0	0	964	1	0	0
4	0.302928	0.070012	0.013331	-0.699130	0.715831	0.816968	0	1	331	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
395	-0.940986	-0.275711	-0.310230	0.031032	-1.373763	-0.144171	0	1	560	0	0	0
396	-0.904963	-0.389362	-0.381413	1.491355	0.541698	1.137347	0	0	480	0	1	0
397	0.359462	-0.244913	-0.219633	1.491355	0.657787	-0.464550	0	1	138	0	0	0
398	-0.212808	-0.958916	-1.054419	-1.429291	-0.677231	-0.144171	0	1	0	0	0	0
399	-0.753345	0.341993	0.388661	1.491355	0.483654	-2.066448	0	0	966	1	0	0

400 rows × 12 columns

There are 400 observations in total $(n = 400)$ with $J = 12$ columns of explanatory variables. Based on the non-linear regression above, this would yield $2 \times (12) + \frac{12!}{2!(12-2)!}$ number of predictors, which is equal to $p = 90$ predictors in total.

An OLS can be used for this, however with so many predictors and it is likely that a reasonable amount of them do not play a large role in the prediction of an individual's credit score, another thing to consider is that given that there are a lot of predictors in comparison to the overall sample size, although it is not an exact science, there would be around 4 observations per predictor, which is somewhat low.

A shrinkage method such as the Lasso can identify these less relevant predictors and shrink their coefficient towards zero which is beneficial in identifying the important predictors, whereas the OLS does not penalise the less relevant predictors, by comparison the Lasso produces a far more succinct model.

(b)

```
In [126]: #Dropping the dependent variable
df2 = df.drop('Balance',axis=1)
```

In [127]:

```
#Transforming the dataframe to conform to the non-linear regression function
#Acquiring the pairwise combination across all regressors
df3 = df2.apply(lambda s:
                   pd.Series(
                       {i: c for i, c in enumerate(combinations_with_replacement(s.values, 2))}
                   ),
                   axis=1)

#Multiplying each pair by one another for all observations across all rows
for k in range(0,66):
  for i in range(0,400):
    df3[k][i]=df3[k][i][0]*df3[k][i][1]

#Generating column names using the combination function again
combs = list(combinations_with_replacement(df2, 2))
colnames = {}
for i in range(0,66):
  colnames[i]=(combs[i][0]+'*'+combs[i][1])

#Applying the new column names to the new dataframe
df3 = df3.rename(columns=dict(colnames))
```

In [128]:

```
#Merge df and add intercept
df4 = pd.concat([df2,df3],axis=1)
df4["intercept"] = 1
```

In [129]:
```
df4
```

Out[129]:

	Income	Limit	Rating	Cards	Age	Education	Student	Married	Asian	African American	Male	Income*Income	Income*Limit	Income*Rating	Income*Cards	I
0	-0.861583	-0.489999	-0.465539	-0.699130	-1.257674	-0.784930	0	1	0	0	0	0.742325	0.422175	0.4011	0.602358	
1	1.727437	0.828261	0.828703	0.031032	1.528451	0.496588	1	1	1	0	0	2.98404	1.43077	1.43153	0.0536056	
2	1.686756	1.014787	1.029311	0.761194	0.889964	-0.784930	0	0	1	0	0	2.84514	1.7117	1.7362	1.28395	
3	2.946152	2.068440	2.110003	0.031032	-1.141586	-0.784930	0	0	1	0	0	8.67981	6.09394	6.21639	0.0914246	
4	0.302928	0.070012	0.013331	-0.699130	0.715831	0.816968	0	1	0	0	0	0.0917652	0.0212086	0.00403824	-0.211786	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
395	-0.940986	-0.275711	-0.310230	0.031032	-1.373763	-0.144171	0	1	0	0	0	0.885455	0.25944	0.291922	-0.0292006	
396	-0.904963	-0.389362	-0.381413	1.491355	0.541698	1.137347	0	0	0	1	0	0.818959	0.352358	0.345165	-1.34962	
397	0.359462	-0.244913	-0.219633	1.491355	0.657787	-0.464550	0	1	0	0	0	0.129213	-0.0880367	-0.0789496	0.536085	
398	-0.212808	-0.958916	-1.054419	-1.429291	-0.677231	-0.144171	0	1	0	0	0	0.0452873	0.204065	0.224389	0.304165	
399	-0.753345	0.341993	0.388661	1.491355	0.483654	-2.066448	0	0	1	0	0	0.567529	-0.257639	-0.292796	-1.1235	

400 rows × 78 columns

In [130]:
```
X = df4
y = df["Balance"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)

clf = Lasso(alpha=0.5,fit_intercept=False,normalize=False,max_iter=5000)
lassomodel = clf.fit(X_train,y_train)
```

In [131]:

```
colnameslist = df4.columns.tolist()
coefficientlist = lassomodel.coef_.tolist()

dictionary = dict(zip(colnameslist, coefficientlist))
output = pd.DataFrame.from_dict(dictionary,orient='index')
output = output.rename(columns={0:"Coef"})
pd.set_option('display.max_rows', 200)
output
```

Out[131]:

	Coef
Income	-279.512419
Limit	568.544133
Rating	35.917929
Cards	19.494562
Age	-11.018457
Education	-1.508624
Student	375.795007
Married	-0.000000
Asian	4.784912
African American	-0.000000
Male	0.000000
Income*Income	59.431296
Income*Limit	-118.414702
Income*Rating	-89.217009
Income*Cards	-6.246934
Income*Age	4.695557
Income*Education	-5.797474
Income*Student	-43.964824
Income*Married	-5.019682
Income*Asian	4.499099
Income*African American	-11.527152
Income*Male	0.000000
Limit*Limit	231.932642
Limit*Rating	0.000000
Limit*Cards	0.000000
Limit*Age	-0.000000
Limit*Education	0.000000
Limit*Student	140.291708
Limit*Married	0.000000
Limit*Asian	0.000000

	Coef
Limit*African American	0.000000
Limit*Male	0.000000
Rating*Rating	-50.664273
Rating*Cards	18.764041
Rating*Age	-9.028662
Rating*Education	1.421649
Rating*Student	0.000000
Rating*Married	0.000000
Rating*Asian	5.592776
Rating*African American	19.533372
Rating*Male	0.000000
Cards*Cards	5.353709
Cards*Age	-2.648918
Cards*Education	-4.308919
Cards*Student	6.146811
Cards*Married	-2.510920
Cards*Asian	0.000000
Cards*African American	0.000000
Cards*Male	0.000000
Age*Age	-3.499872
Age*Education	1.241653
Age*Student	0.000000
Age*Married	-8.831830
Age*Asian	11.466764
Age*African American	-0.000000
Age*Male	0.000000
Education*Education	-2.478272
Education*Student	28.866734
Education*Married	-4.282505
Education*Asian	-11.281416
Education*African American	-2.552950

	Coef
Education*Male	0.000000
Student*Student	57.993774
Student*Married	0.956165
Student*Asian	-0.000000
Student*African American	0.000000
Student*Male	0.000000
Married*Married	-0.000000
Married*Asian	-0.000000
Married*African American	0.388309
Married*Male	0.000000
Asian*Asian	0.000000
Asian*African American	0.000000
Asian*Male	0.000000
African American*African American	-0.000000
African American*Male	0.000000
Male*Male	0.000000
intercept	399.530522

In [132]:
```
pd.set_option('display.max_rows', 25)
```

variables like student, income, and rating may have a better explanation when it comes to credit card debt. Students are most likely to be vulnerable to credit debt because of their spending habits and lack of income, we can also see that people with more education are more likely to have less debt may be due to higher incomes facilitated by their extra qualifications. Also, an indication with age, whereby as people grow older they tend to spend less which could lower their credit debt.

(c)

In [133]:
```
#Training inputs being used to predict outputs
ypred = clf.predict(X_train)
```

In [134]:
```
mean_squared_error(y_train, ypred)
```

Out[134]: 2275.343346090848

(d)

In [135]:
```
#Making cross-validation predictions
clfcv = LassoCV(cv=KFold(5), random_state=0, max_iter=100000)
abs(cross_val_score(clfcv, X, y, scoring='neg_mean_squared_error').mean())
```

Out[135]: 3874.9378056555565

(e)

We estimate the training MSE to be 2275 while the 5-fold CV gives us an estimation of 2737. The discrepancy could occur here due to a difference in the two methods. The Training MSE is estimated through the average of all the squared differences between the dependent variable and the predicted variables. We expect the training MSE to be small because the data used is the same data that was used to train the model so the predicted variables are close to the true responses, which causes overfitting. Compared to Using the 5-fold CV which holds out each fold for testing purposes and reduces this overfitting problem. To compute the 5-fold CV we randomly split the data sample into 5 folds, the first fold out of the 5 is the validation set. The 5-fold CV MSE is derived from an average MSE which is calculated on each fold, so we take the average of the 5 MSE's.

(f)

$i.$

For λ, we will use 100 values between 0 and 0.5, with equal intervals this increments in steps of 0.005.

In [136]:
```
#List of lambda values from 0.01 to 0.99
lambdalist = (np.array(list(range(1,10000,100))))/1000
(lambdalist)
```

Out[136]:
```
array([1.000e-03, 1.010e-01, 2.010e-01, 3.010e-01, 4.010e-01, 5.010e-01,
       6.010e-01, 7.010e-01, 8.010e-01, 9.010e-01, 1.001e+00, 1.101e+00,
       1.201e+00, 1.301e+00, 1.401e+00, 1.501e+00, 1.601e+00, 1.701e+00,
       1.801e+00, 1.901e+00, 2.001e+00, 2.101e+00, 2.201e+00, 2.301e+00,
       2.401e+00, 2.501e+00, 2.601e+00, 2.701e+00, 2.801e+00, 2.901e+00,
       3.001e+00, 3.101e+00, 3.201e+00, 3.301e+00, 3.401e+00, 3.501e+00,
       3.601e+00, 3.701e+00, 3.801e+00, 3.901e+00, 4.001e+00, 4.101e+00,
       4.201e+00, 4.301e+00, 4.401e+00, 4.501e+00, 4.601e+00, 4.701e+00,
       4.801e+00, 4.901e+00, 5.001e+00, 5.101e+00, 5.201e+00, 5.301e+00,
       5.401e+00, 5.501e+00, 5.601e+00, 5.701e+00, 5.801e+00, 5.901e+00,
       6.001e+00, 6.101e+00, 6.201e+00, 6.301e+00, 6.401e+00, 6.501e+00,
       6.601e+00, 6.701e+00, 6.801e+00, 6.901e+00, 7.001e+00, 7.101e+00,
       7.201e+00, 7.301e+00, 7.401e+00, 7.501e+00, 7.601e+00, 7.701e+00,
       7.801e+00, 7.901e+00, 8.001e+00, 8.101e+00, 8.201e+00, 8.301e+00,
       8.401e+00, 8.501e+00, 8.601e+00, 8.701e+00, 8.801e+00, 8.901e+00,
       9.001e+00, 9.101e+00, 9.201e+00, 9.301e+00, 9.401e+00, 9.501e+00,
       9.601e+00, 9.701e+00, 9.801e+00, 9.901e+00])
```

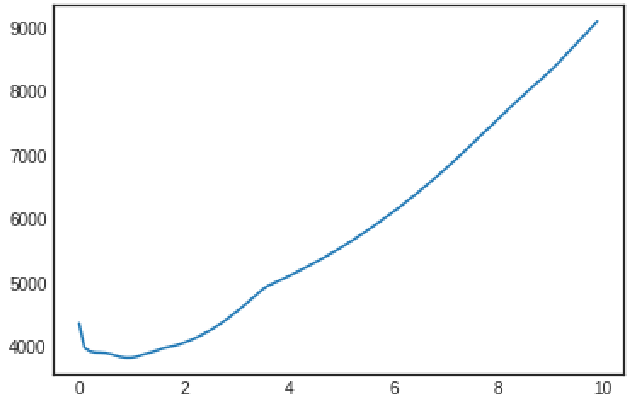
In [137]:
```
#CVn = []
#for i in lambdalist:
  #clf = gln.ElasticNet(alpha=1, lambda_path=lambdalist, scoring='mean_squared_error', n_splits=5,max_iter=1000)
  #CVn.append(mean_squared_error(y,clf.fit(X,y).predict(X)))
```

In [138]:
```
CVn = []
for i in lambdalist:
  lasso = Lasso(alpha=i, fit_intercept=False,max_iter=10000,tol=0.001)
  CVn.append(abs(cross_val_score(lasso, X, y, cv=5, scoring='neg_mean_squared_error').mean()))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not converge. You m
ight want to increase the number of iterations. Duality gap: 285026.8018716332, tolerance: 152407.818
  positive)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not converge. You m
ight want to increase the number of iterations. Duality gap: 201454.96481441538, tolerance: 154580.269
  positive)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not converge. You m
ight want to increase the number of iterations. Duality gap: 331365.6151270569, tolerance: 149719.74
  positive)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not converge. You m
ight want to increase the number of iterations. Duality gap: 210821.82172418764, tolerance: 163596.83800000002
  positive)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not converge. You m
ight want to increase the number of iterations. Duality gap: 275631.600792871, tolerance: 149719.943
  positive)
```

In [139]:
```
sns.lineplot(x=lambdalist,y=CVn)
```

Out[139]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb89909da90>
```



In [140]:
```
CVn = pd.DataFrame(CVn)
lambdalist = pd.DataFrame(lambdalist)
pd.concat([CVn,lambdalist],axis=1)
#the MSE is minimised at alpha = 1, which implies that lambda = 0.901
```

Out[140]:

	0	0
0	4364.660580	0.001
1	3981.941359	0.101
2	3927.352941	0.201
3	3906.380523	0.301
4	3904.453208	0.401
...	...	...
95	8751.832999	9.501
96	8837.310061	9.601
97	8923.539941	9.701
98	9010.886489	9.801
99	9099.738838	9.901

100 rows × 2 columns

$ii.$

In [141]:

```
X = df4
y = df["Balance"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)

clf = Lasso(alpha=0.901,fit_intercept=False,normalize=False,max_iter=5000)
lassomodel = clf.fit(X_train,y_train)
lassomodel.coef_

colnameslist = df4.columns.tolist()
coefficientlist = lassomodel.coef_.tolist()

dictionary = dict(zip(colnameslist, coefficientlist))
output = pd.DataFrame.from_dict(dictionary,orient='index')
output = output.rename(columns={0:"Coef"})
pd.set_option('display.max_rows', 200)
max_abs_diff = 0.00001
output
```

Out[141]:

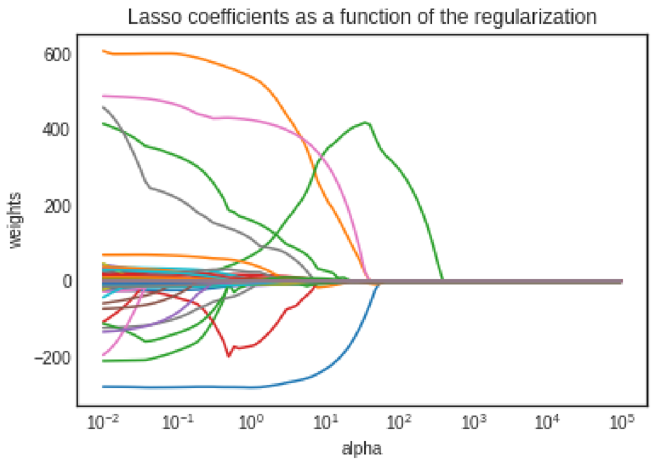
	Coef
Income	-280.796847
Limit	547.533418
Rating	59.013250
Cards	17.668327
Age	-10.845343
Education	-2.083350
Student	337.402461
Married	-0.000000
Asian	3.916612
African American	-0.000000
Male	0.000000
Income*Income	51.111418
Income*Limit	-19.115994
Income*Rating	-167.942447
Income*Cards	-0.633350
Income*Age	1.073679
Income*Education	-3.882616
Income*Student	-22.204511
Income*Married	-3.316764
Income*Asian	0.000000
Income*African American	-0.189039
Income*Male	0.000000
Limit*Limit	168.201337
Limit*Rating	0.000000
Limit*Cards	0.228222
Limit*Age	-0.000000
Limit*Education	0.377283
Limit*Student	117.204717
Limit*Married	-0.000000
Limit*Asian	0.000000

	Coef
Limit*African American	0.000000
Limit*Male	0.000000
Rating*Rating	-0.000000
Rating*Cards	11.543375
Rating*Age	-6.004909
Rating*Education	0.000000
Rating*Student	0.000000
Rating*Married	0.000000
Rating*Asian	5.095282
Rating*African American	4.806905
Rating*Male	0.000000
Cards*Cards	5.798753
Cards*Age	-3.294416
Cards*Education	-4.540173
Cards*Student	0.604883
Cards*Married	-0.390167
Cards*Asian	0.000000
Cards*African American	0.000000
Cards*Male	0.000000
Age*Age	-2.883222
Age*Education	0.000000
Age*Student	-0.000000
Age*Married	-7.678574
Age*Asian	7.875136
Age*African American	-0.000000
Age*Male	0.000000
Education*Education	-1.757459
Education*Student	25.530952
Education*Married	-5.251801
Education*Asian	-7.792798
Education*African American	-0.000000

	Coef
Education*Male	0.000000
Student*Student	94.521178
Student*Married	0.000000
Student*Asian	0.000000
Student*African American	0.000000
Student*Male	0.000000
Married*Married	-0.000000
Married*Asian	-0.000000
Married*African American	0.000000
Married*Male	0.000000
Asian*Asian	0.681410
Asian*African American	0.000000
Asian*Male	0.000000
African American*African American	-0.000000
African American*Male	0.000000
Male*Male	0.000000
intercept	402.520278

$iii.$

In [142]:
```
alphas = 10**np.linspace(5,-2,100)*0.5
lasso = Lasso(max_iter=100000)
coefs = []

for a in alphas*2:
    lasso.set_params(alpha=a)
    lasso.fit(X_train, y_train)
    coefs.append(lasso.coef_)

ax = plt.gca()
ax.plot(alphas*2, coefs)
ax.set_xscale('log')
plt.axis('tight')
plt.xlabel('alpha')
plt.ylabel('weights')
plt.title('Lasso coefficients as a function of the regularization');
plt.rcParams["figure.figsize"] = (10,10)
```



iv.

In [143]:
```
#Out of Sample
OOS = (pd.DataFrame.from_dict({'Income':100,'Limit':6000,'Rating':500,'Cards':3,'Age':70,'Education':12,'Student':0,'Married':1,'Asian':1,'Af
rican American':0,'Male':0},orient='index')).T
```

Question (2)

(a)

In [144]:
```
# dummy variables
#credit = pd.read_csv('Credit.csv')
#credit.dropna()
#credit.drop("ID",axis=1,inplace=True)
#credit['Female'] = credit.Gender.map({' Male':0, 'Female':1})
#credit['Student'] = credit.Student.map({'No':0, 'Yes':1})
#credit['Married'] = credit.Married.map({'No':0, 'Yes':1})
##Dummy for Ethnicity
#credit.loc[:,"Asian"] = [1 if x=="Asian" else 0 for x in credit["Ethnicity"]]
#credit.loc[:,"African American"] = [1 if x=="African American" else 0 for x in credit["Ethnicity"]]
#credit.drop(columns="Ethnicity",inplace=True)
#credit.info()
```

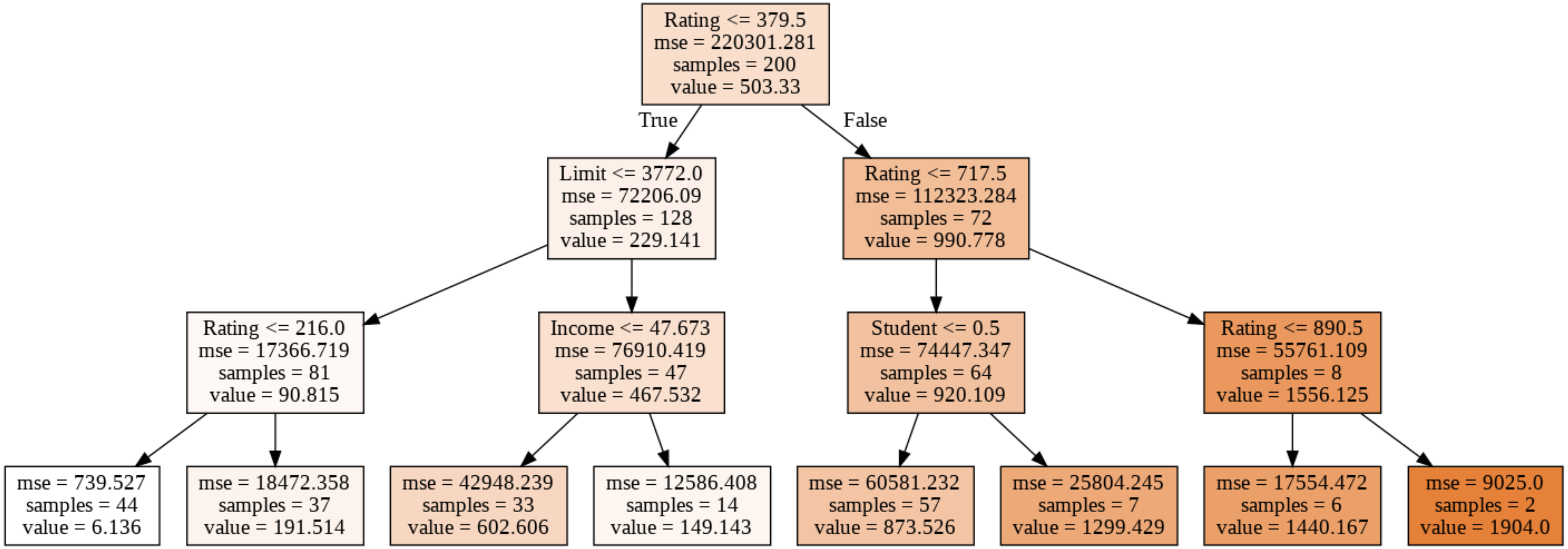
In [145]:
```
def print_tree(estimator, features, class_names=None, filled=True):
    tree = estimator
    names = features
    color = filled
    classn = class_names

    dot_data = StringIO()
    export_graphviz(estimator, out_file=dot_data, feature_names=features, class_names=classn, filled=filled)
    graph = pydot.graph_from_dot_data(dot_data.getvalue())
    return(graph)
```

In [146]:
```
df = pd.read_csv("Credit.csv")

#student, married, ethnicity

#Create dummies for:
df.drop("ID",axis=1,inplace=True)
df.dropna()

#Dummy for Student
df.loc[:,"Student"] = [1 if x=="Yes" else 0 for x in df["Student"]]

#Dummy for Married
df.loc[:,"Married"] = [1 if x=="Yes" else 0 for x in df["Married"]]

#Dummy for Ethnicity
df.loc[:,"Asian"] = [1 if x=="Asian" else 0 for x in df["Ethnicity"]]
df.loc[:,"African American"] = [1 if x=="African American" else 0 for x in df["Ethnicity"]]
df.drop(columns="Ethnicity",inplace=True)

#Dummy for Gender
df.loc[:,"Male"] = [1 if x=="Male" else 0 for x in df["Gender"]]
df.drop(columns="Gender",inplace=True)

X = df.drop('Balance', axis=1)
y = df.Balance

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
```

In [147]:
```
# the tree regression
regr = DecisionTreeRegressor(max_depth=3,)
regr.fit(X_train, y_train)
pred = regr.predict(X_test)
```

In [148]:
```
# visualize the tree
graph, = print_tree(regr, features=X.columns)
Image(graph.create_png())
```

Out[148]:



In [148]:

The source node at the top implies that rating is the most important variable in determining an individuals debt balance. The right hand side branches into assigning those with higher levels of debt and the left hand side less so. It also shows that being a student has a strong impact on the predicted balance of the individual. On the other hand income also plays a larger role on the left hand side branch which assigns those on a lower level of income to a higher level of debt, which is a plausible conclusion to draw. The issue with this tree however, is that with such a range of possible balances, there are so few terminal nodes, which consequently causes a very large MSE.

(b)

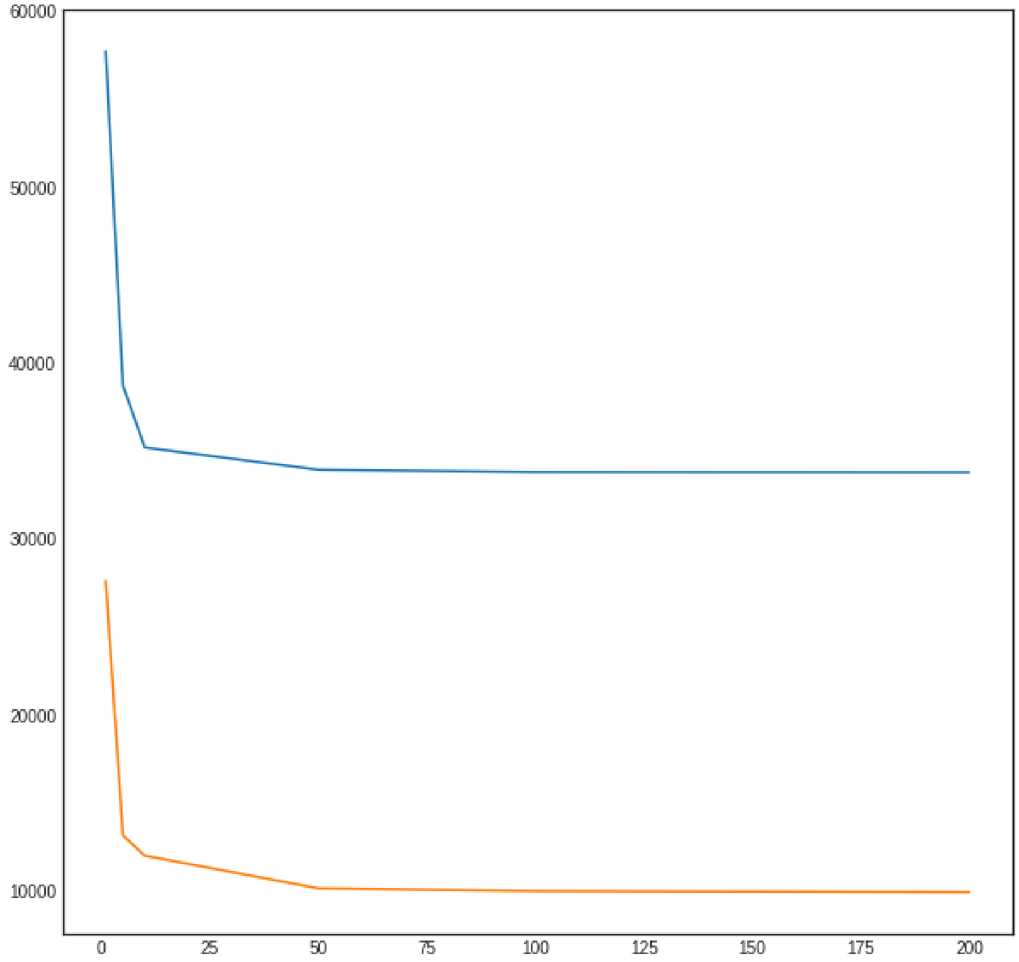
In [149]:
```
#CV for Test MSE
abs(cross_val_score(regr, X, y, cv=5, scoring='neg_mean_squared_error').mean())
```

Out[149]: 52043.713781530685

(c)

In [150]:
```
trees = [1, 5, 10, 50, 100, 200]
testmse_3depth = []


for i in trees:
  randomforest = RandomForestRegressor(n_estimators=i,max_depth=3, random_state=0)
  testmse_3depth.append(abs(cross_val_score(randomforest, X, y, cv=5, scoring='neg_mean_squared_error').mean()))

testmse_muchdepth = []
for i in trees:
  randomforest = RandomForestRegressor(n_estimators=i, random_state=0)
  testmse_muchdepth.append(abs(cross_val_score(randomforest, X, y, cv=5, scoring='neg_mean_squared_error').mean()))

sns.lineplot(x=trees,y=testmse_3depth)
sns.lineplot(x=trees,y=testmse_muchdepth)
```

Out[150]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb895efb910>



The clearest finding from the plot above is that using the forest which does not limit the depth results in a far more reduced test MSE, this is because there is more flexibility in capturing information about the input data and subsequent predictions are then likely to yield more accurate results. Although it is important that we do not go too deep and end up overfitting the data. That would not be ideal.

(d)

From the prior section, using the 50 tree random forest on the unrestricted depth yields the lowest test MSE using the 5 fold CV, since it is almost identical in test MSE to those with a larger number of trees it may be better to use a more parsimonious model to perform the out of sample prediction.

In [151]:
```
#Out of sample obs
OOS
```

Out[151]:

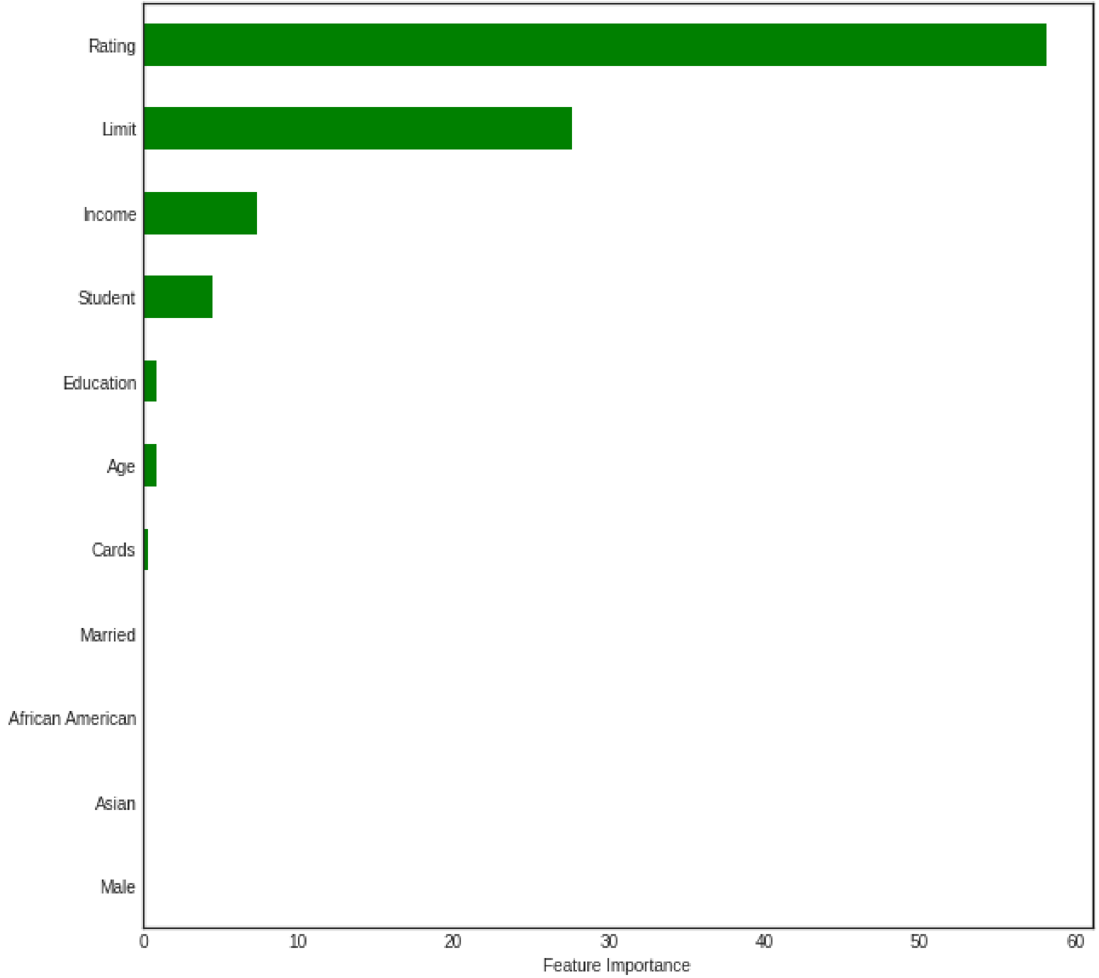
	Income	Limit	Rating	Cards	Age	Education	Student	Married	Asian	African American	Male
0	100	6000	500	3	70	12	0	1	1	0	0

In [152]:
```
rf50 = RandomForestRegressor(n_estimators=50, random_state=0)
prediction = rf50.fit(X_train,y_train).predict(OOS)
prediction
```

Out[152]:
```
array([807.92])
```

The prediction for the stated individual would have an outstanding debt balance of around 808.

In [153]:
```
importances = rf50.feature_importances_*100

imp = pd.Series(importances,index=X.columns.tolist()).sort_values(inplace=False)
imp.T.plot(kind='barh', color='g', figsize=(10,10))

plt.xlabel('Feature Importance')
plt.gca().legend_ = None
```



Again we see that using a random forest rating plays the most important role in determining the balance of an individual, which is closely followed by limit and income, this follows quite closely with the very first tree seen in part (a).