In [212]:
```
%matplotlib inline
import matplotlib as mlp
import scipy
import numpy as np
import sympy
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import scale
from scipy.stats.stats import spearmanr
from sklearn.linear_model import LinearRegression
from sklearn import cross_validation
from sklearn import metrics
import scipy.sparse as sps
import math
from bs4 import BeautifulSoup
import requests
import re
import fileinput
import math as mt
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import datasets, linear_model

df = pd.read_csv('Merged_Dataset_new.csv');
```

#TODO:

1. Normalise total star votes, stars. Make a weighted book score. Add and find correlation
2. Check correlation of genres. If anything interesting we will think about grouping/ scoring based on genre
3. Baseline - Linear
4. Advanced - Linear with regularization, Logistic, K-means approach.
5. Final prediction for a set of new books (TBD Sayan)

In [213]:
```
df = df.drop('Unnamed: 0', axis=1)
# df = df.drop_duplicates(subset=['New Book Name'], keep='last')
# df = df.set_index([range(df['Link'].values.size)])
```

In [214]:
```
print df.columns
```

```
Index([u'New Book Name', u'ID', u'Link', u'Budget', u'Gross', u'Opening',
       u'Release Date', u'Awards List', u'Book Name', u'Characters', u'Ge
nres',
       u'Num_reviews', u'Other editions', u'Pages', u'Star votes', u'Star
s',
       u'publish_date', u'Award Count', u'Num Characters', u'Star_count_
5',
       u'Star_count_4', u'Star_count_3', u'Star_count_2', u'Star_count_
1',
       u'Star_count_total', u'Genre_Fiction', u'Genre_Fantasy',
       u'Genre_Classics', u'Genre_Mystery', u'Genre_Young Adult',
       u'Genre_Romance', u'Genre_Literature', u'Genre_Historical Fictio
n',
       u'Genre_Thriller', u'Genre_Childrens', u'Genre_Science Fiction',
       u'Genre_Novels', u'Genre_Contemporary', u'Genre_Adventure',
       u'Genre_Crime', u'Genre_Cultural', u'Genre_Sequential Art',
       u'Genre_Historical', u'Genre_Humor', u'Genre_Horror',
       u'Genre_Paranormal', u'Genre_Nonfiction', u'Genre_War',
       u'Genre_European Literature', u'Genre_Biography',
       u'Genre_Womens Fiction', u'Genre_Suspense', u'Genre_Chick Lit',
       u'Genre_Adult', u'Genre_Autobiography', u'Genre_History', u'Time D
iff'],
      dtype='object')
```

In [215]:
```
df['Budget'] = df['Budget'].astype(str).map(lambda x: float(x[3:])*0.74 if '
#df['Budget'] = df['Budget'].fillna(df['Budget'].mean())

df['Gross'] = df['Gross'].astype(str).map(lambda x: float(x[3:])*0.74 if 'CA
#df['Gross'] = df['Gross'].fillna(df['Gross'].mean())

df['Opening'] = df['Opening'].astype(str).map(lambda x: float(x[3:])*0.74 if
#df['Opening'] = df['Opening'].fillna(df['Opening'].mean())
```

In [216]:

```
def budget_score_movie(row):
    if not mt.isnan(row['Gross']) and not mt.isnan(row['Budget']):
        return np.log((row['Gross']-row['Budget'])/row['Budget'])
    else:
        return np.NaN

def budget_score_movie_nonlog(row):
    if not mt.isnan(row['Gross']) and not mt.isnan(row['Budget']):
        return (1 if ((row['Gross']-row['Budget'])/row['Budget']) > 1.5 e
    else:
        return np.NaN


df['Score'] = df.apply(budget_score_movie,axis=1)

df['Score binary'] = df.apply(budget_score_movie_nonlog,axis=1)

print "Number of non nan scores: ",df['Score'].map(lambda x: not mt.isnar

#Lets just take the finite values for now
df = df[np.isfinite(df['Score'])]

final_df = df[['Score','Score binary','Budget','Gross','Opening','Num Cha
               'Other editions','Pages','Star_count_5','Star_count_4','St
              'Star_count_total','Time Diff','Genre_Fiction', 'Genre_Fant
               'Genre_Romance', 'Genre_Literature', 'Genre_Historical Fic
               'Genre_Science Fiction', 'Genre_Novels', 'Genre_Contempora
               'Genre_Suspense', 'Genre_Chick Lit', 'Genre_Adult', 'Genre

#final_df['Time Diff'] = pd.to_datetime(final_df['Time Diff'], coerce=Tr

final_df['Time Diff'] = final_df['Time Diff'].astype('str').map(lambda x

# normed_Star_count_total = np.array((final_df['Star_count_total'] - fina
#                   final_df['Star_count_total'].std(ddof=0))
# normed_Stars = np.array((final_df['Stars'] - final_df['Stars'].mean())
#                   final_df['Stars'].std(ddof=0))

normed_Star_count_total = np.log((final_df['Star_count_total']))
normed_Stars = np.log((final_df['Stars']))

final_df['Book score'] = normed_Star_count_total + normed_Stars

print final_df['Book score']
# Check what all are int or float
# print final_df.applymap(lambda x: isinstance(x, (int, float))).all(0)
# print df.applymap(lambda x: isinstance(x, (int, float))).all(0)
for i in final_df.columns.values:
    final_df[i] = final_df[i].fillna(final_df[i].mean())
# final_df
```

```
Number of non nan scores:  364
```

```
/usr/local/lib/python2.7/site-packages/ipykernel/__main__.py:32: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.or
g/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
/usr/local/lib/python2.7/site-packages/ipykernel/__main__.py:42: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.or
g/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
/usr/local/lib/python2.7/site-packages/ipykernel/__main__.py:49: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.or
g/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
```

1	13.091962
3	9.418891
4	13.432542
5	11.905445
8	12.839755
12	14.351092
22	14.100682
24	12.084809
28	13.032877
31	11.098120
32	13.521807
35	14.660773
38	14.621084
40	12.929954
43	12.594267
49	13.293360
50	14.238724
52	10.896904
53	6.563884
55	13.742141
57	13.993988
59	7.815728
60	13.350972
74	9.912848
76	9.566552
78	13.153910
85	14.114869
89	8.656816
92	8.439526
97	9.787016
...	...
1145	8.009523
1147	16.394073
1148	10.555342
1150	12.955760
1151	8.144560
1152	7.074252
1155	11.559066
1157	13.422994
1159	16.382273
1162	13.813458
1164	8.815095
1165	7.820375
1166	7.066467
1169	9.348365
1170	13.714792
1171	11.727432
1176	7.764805
1177	10.739311
1180	13.471762
1181	8.779477
1182	12.759563
1186	15.262317
1192	11.846067
1193	13.604974
1197	12.935788
1199	13.751423

```
1201     8.110400
1203    11.331145
1211    10.273384
1212    10.397192
Name: Book score, dtype: float64
```

In [217]:
```
corr_mat = final_df.corr(method="pearson")
corr_mat
```

Out[217]:

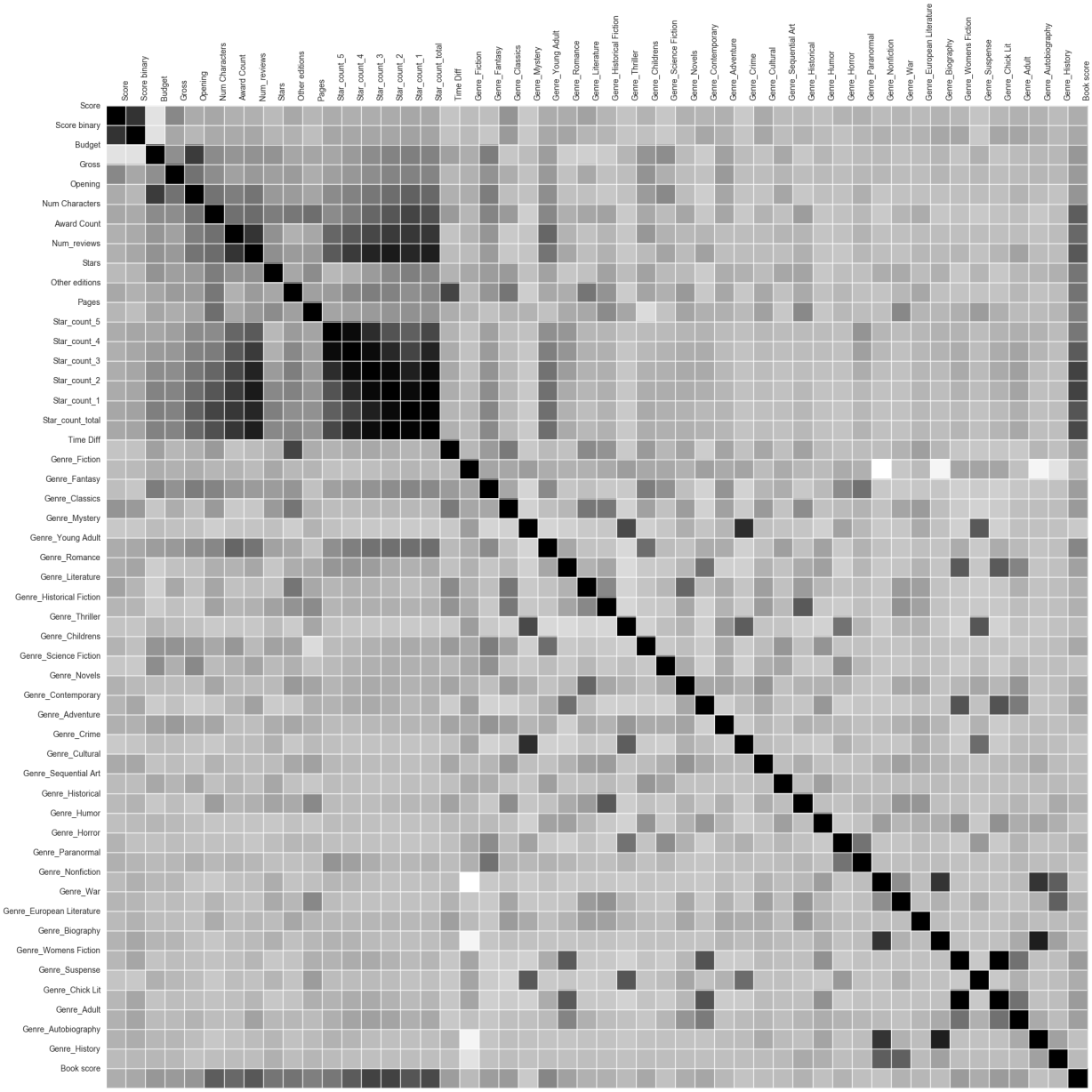
	Score	Score binary	Budget	Gross	Opening	Num Characters	Av Co
Score	1.000000	0.724785	-0.327677	0.251413	0.054669	0.077924	0.(
Score binary	0.724785	1.000000	-0.320682	0.078549	-0.004578	0.065467	0.(
Budget	-0.327677	-0.320682	1.000000	0.207002	0.688214	0.231792	0.1
Gross	0.251413	0.078549	0.207002	1.000000	0.378301	0.220596	0.1
Opening	0.054669	-0.004578	0.688214	0.378301	1.000000	0.375961	0.3
Num Characters	0.077924	0.065467	0.231792	0.220596	0.375961	1.000000	0.3
Award Count	0.028402	0.062396	0.171230	0.110505	0.312332	0.392066	1.(
Num_reviews	0.033880	0.079971	0.193051	0.157057	0.371248	0.421069	0.7
Stars	-0.020249	0.007228	0.189237	0.124683	0.154117	0.313967	0.1
Other editions	0.067947	0.022209	0.085579	0.144435	0.143242	0.350038	0.(
Pages	-0.003528	-0.049385	0.088826	0.043353	0.091351	0.408029	0.(
Star_count_5	0.052129	0.083331	0.083314	0.122152	0.229293	0.234510	0.4
Star_count_4	0.040601	0.078533	0.148880	0.162007	0.283730	0.323453	0.5
Star_count_3	0.045554	0.081476	0.224346	0.221013	0.350973	0.455552	0.6
Star_count_2	0.057414	0.083636	0.278147	0.273437	0.414694	0.558604	0.6
Star_count_1	0.077450	0.101255	0.305908	0.284777	0.482787	0.643952	0.7
Star_count_total	0.068040	0.096847	0.284130	0.273879	0.448053	0.591028	0.7
Time Diff	-0.088821	-0.077635	0.117253	-0.004373	0.029417	0.149932	-0.
Genre_Fiction	-0.030142	-0.022901	0.046488	0.022856	0.045816	0.000268	0.(
Genre_Fantasy	-0.044200	-0.105109	0.319657	0.192534	0.317197	0.271028	0.1
Genre_Classics	0.179591	0.148596	-0.105083	0.041369	-0.045643	0.147440	-0.
Genre_Mystery	-0.126744	-0.123890	-0.035603	-0.063773	-0.068737	-0.163548	-0.
Genre_Young Adult	0.048060	0.069195	0.127189	0.152712	0.209813	0.255157	0.4
Genre_Romance	0.053074	0.092467	-0.158623	-0.065672	-0.059361	0.037798	0.(
Genre_Literature	0.100842	0.004705	-0.171583	0.067619	-0.061176	0.086282	-0.
Genre_Historical Fiction	0.008629	0.008497	-0.143853	-0.060410	-0.114140	0.101399	0.(
Genre_Thriller	-0.085292	-0.100790	0.005121	-0.037507	-0.069695	-0.128560	-0.
Genre_Childrens	0.011201	-0.057021	0.182048	0.185517	0.143439	0.072684	0.1

	Score	Score binary	Budget	Gross	Opening	Num Characters	Av Co
Genre_Science Fiction	-0.102082	-0.131096	0.216299	0.037022	0.244018	-0.027489	0.(
Genre_Novels	0.025176	-0.018110	-0.027149	-0.038458	-0.038317	0.083035	-0.
Genre_Contemporary	0.004010	0.077094	-0.142123	-0.069409	-0.155202	-0.110838	0.(
Genre_Adventure	0.031111	0.023576	0.106391	0.150224	0.086114	0.085068	-0.
Genre_Crime	-0.130243	-0.099299	-0.033903	-0.048986	-0.085009	-0.118170	-0.
Genre_Cultural	0.063983	0.077757	-0.075653	-0.041787	-0.084758	0.010057	-0.
Genre_Sequential Art	-0.028798	-0.039154	0.069254	0.003052	0.081623	-0.060173	0.(
Genre_Historical	-0.020582	0.013197	-0.111031	-0.041608	-0.110669	0.126254	0.(
Genre_Humor	-0.050384	-0.026039	-0.001809	-0.021764	-0.054083	-0.095873	-0.
Genre_Horror	0.014805	-0.072777	-0.068596	-0.037988	-0.075311	-0.073488	-0.
Genre_Paranormal	0.033648	0.004523	-0.021406	-0.009983	0.035322	-0.016235	0.(
Genre_Nonfiction	-0.025633	0.033230	-0.016111	-0.008914	-0.096620	-0.127494	-0.
Genre_War	0.005757	-0.002809	-0.053262	-0.035271	-0.071237	0.002675	-0.
Genre_European Literature	-0.019405	0.017750	-0.036058	-0.037309	-0.058589	0.047990	-0.
Genre_Biography	0.010058	0.078177	0.008164	0.008107	-0.061421	-0.107700	-0.
Genre_Womens Fiction	-0.007183	0.084856	-0.083761	-0.037125	-0.053999	-0.081194	-0.
Genre_Suspense	-0.091730	-0.120885	0.032494	-0.011394	-0.041227	-0.119355	-0.
Genre_Chick Lit	-0.007183	0.084856	-0.083761	-0.037125	-0.053999	-0.081194	-0.
Genre_Adult	0.024718	0.087737	-0.026247	-0.019477	0.013104	-0.041344	-0.
Genre_Autobiography	0.017613	0.047282	-0.009934	0.000422	-0.042465	-0.109930	-0.
Genre_History	-0.028732	-0.013222	-0.010173	-0.013351	-0.076964	-0.049160	-0.
Book score	0.046586	0.071597	0.159453	0.147405	0.172353	0.512399	0.4

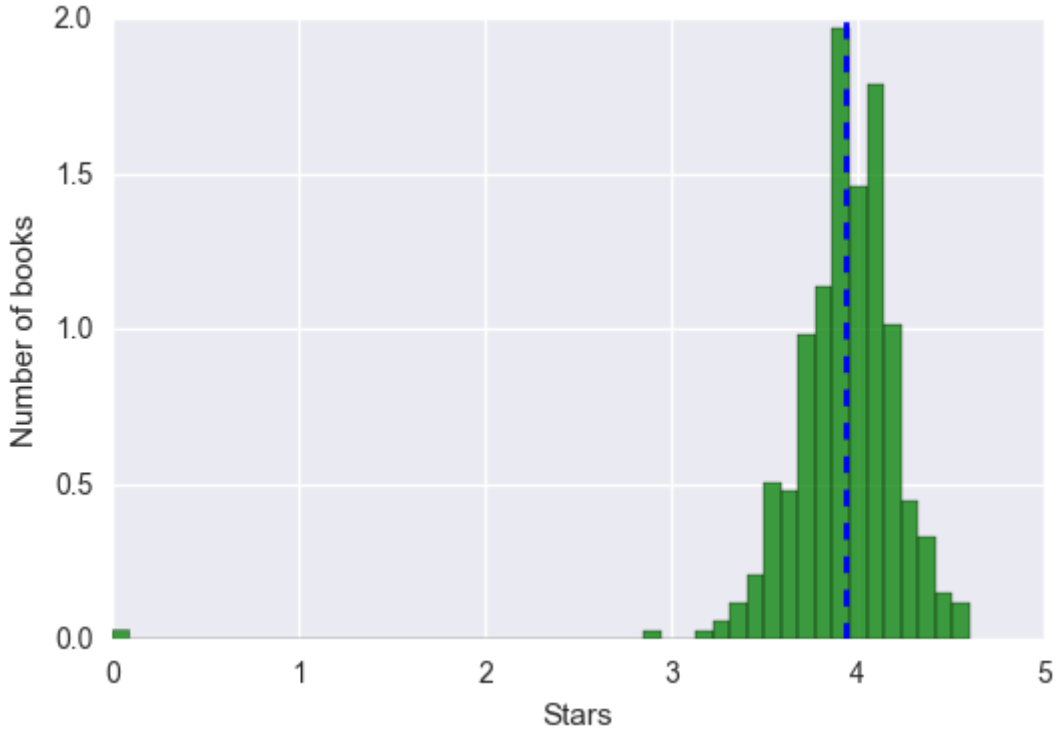
50 rows × 50 columns

In [218]:

```
fig, ax = plt.subplots(figsize=(22, 22))
ax.matshow(corr_mat,extent=[len(corr_mat.columns), 0 , len(corr_mat.columns)
plt.xticks(range(len(corr_mat.columns)+1), list(reversed(corr_mat.columns)),
plt.yticks(range(len(corr_mat.columns)+1), corr_mat.columns);
# plt.show()
```



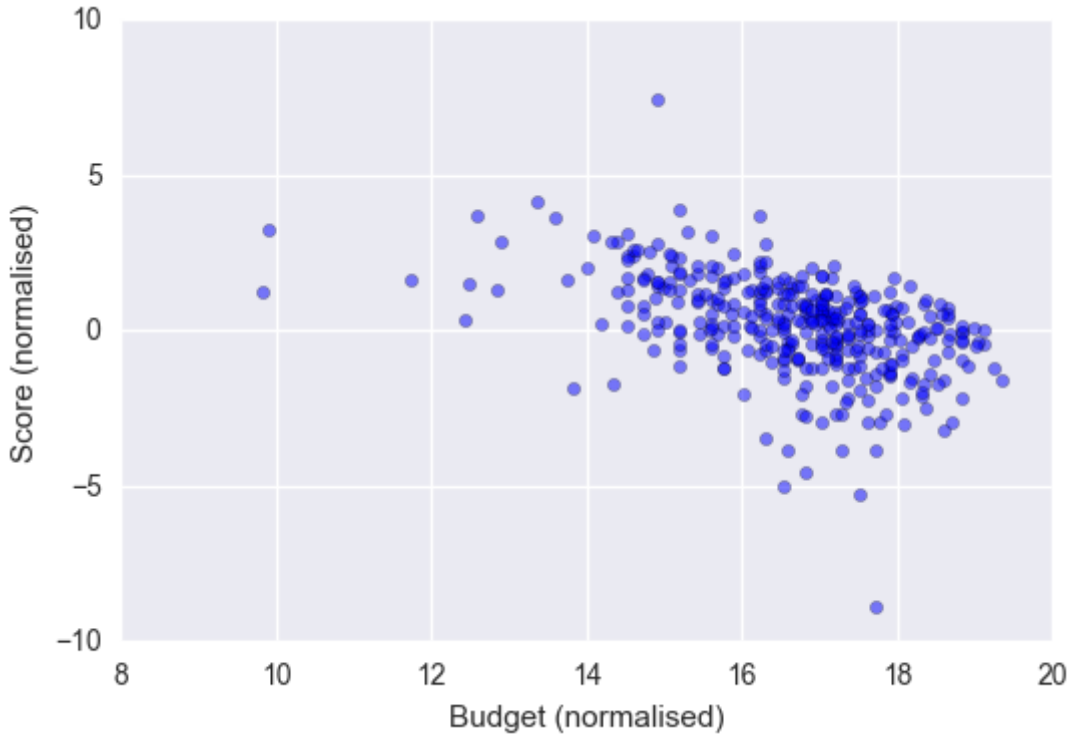
In [219]:
```
# normed_stars = np.array((final_df['Stars'] - final_df['Stars'].mean()) /
#                         final_df['Stars'].std(ddof=0))

normed_stars = np.array((final_df['Stars']))
fig1 = plt.figure()
ax1 = fig1.add_subplot(111)
ax1.hist(normed_stars, 50, normed=1, facecolor='green', alpha=0.75)
ax1.set_xlabel('Stars')
ax1.set_ylabel('Number of books')
ax1.axvline(normed_stars.mean(), color='b', linestyle='dashed', linewidth=2)
ax1.grid(True)
```



From the histogram above we can see that ->

1 Stars almost follow a normal-like distribution
2 We need to clean data much better. We can see outliers to the left of the histogram. We need to ensure that we extrapolate values much better.

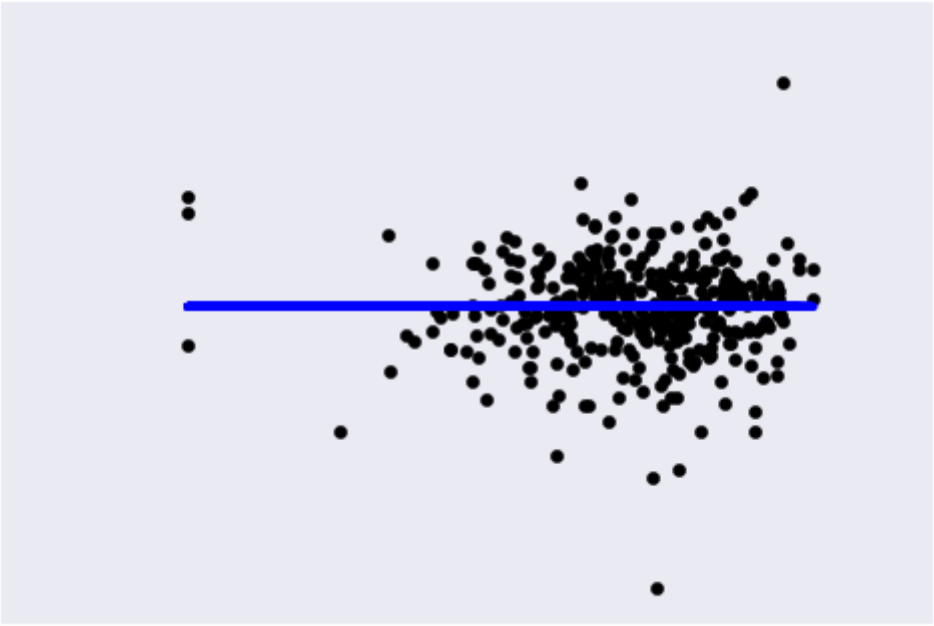
In [220]:
```
####################
#create Scatter plots
####################
## Literacy  vs  Labor Force

# normed_budget = np.array((final_df['Budget'] - final_df['Budget'].mean())
#                    final_df['Budget'].std(ddof=0))

# normed_score = np.array((final_df['Score'] - final_df['Score'].mean()) /
#                    final_df['Score'].std(ddof=0))
#create scatter plot for each country


normed_budget = np.log((final_df['Budget']))
normed_score = np.array((final_df['Score']))

x = normed_budget
y = normed_score


plt.scatter(x, y, alpha=0.5)

plt.xlabel("Budget (normalised)")
plt.ylabel("Score (normalised)")
plt.show()
```



A graph of budget vs profit/budget can be very insightful. We can understand whether the profit a movie earns really depends upon the initial budget or not. From here, we can see that apart from outliers, most of the movies are in a cluster where budget is nearly equal to score. However, the number of outliers is not negligible so it is evident that something more from the money drives sales.

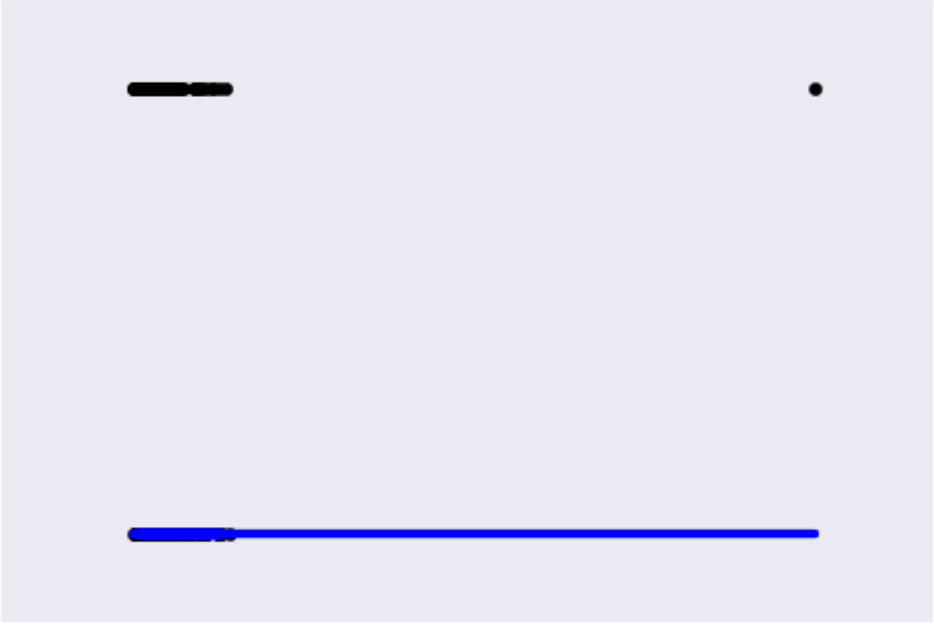
In [221]:
```
normed_book_score = np.array((final_df['Book score']))

print "Number of non nan scores: ",final_df['Book score'].map(lambda x: not
print "len of df : ", len(final_df)

y = normed_score.reshape(len(final_df), 1)
x = normed_book_score.reshape(len(final_df), 1)

# print np.shape(x)
# print np.shape(y)
# print np.isfinite(x).sum()
y[np.isnan(y)] = 0
x[np.isneginf(x)] = 0
# print np.isfinite(y).sum()
# print np.isfinite(x).sum()

regr = linear_model.LinearRegression()
regr.fit(x, y)
plt.scatter(x, y,  color='black')
plt.plot(x, regr.predict(x), color='blue', linewidth=3)
plt.xticks(())
plt.yticks(())
plt.show()
```

```
Number of non nan scores:  364
len of df :  364
```



In [222]:
```
# normed_book_score = np.array((final_df['Book score']))

normed_book_score = np.array((final_df.sum(axis=1)))

# print normed_book_score

print "Number of non nan scores: ",final_df['Book score'].map(lambda x: not
print "len of df : ", len(final_df)

y = final_df['Score binary'].reshape(len(final_df), 1)
x = normed_book_score.reshape(len(final_df), 1)


print np.shape(x)
print np.shape(y)
print np.isfinite(x).sum()
y[np.isnan(y)] = 0
x[np.isneginf(x)] = 0
print np.isfinite(y).sum()
print np.isfinite(x).sum()

logis = linear_model.LogisticRegression()
logis.fit(x, y)
plt.scatter(x, y,  color='black')
plt.plot(x, logis.predict(x), color='blue', linewidth=3)
plt.xticks(())
plt.yticks(())
plt.show()
```

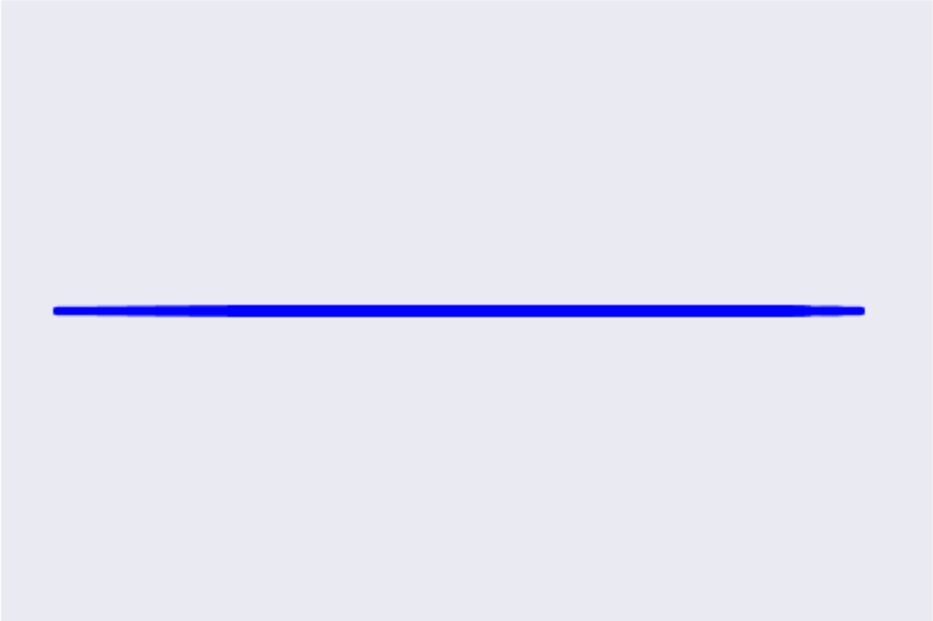
```
Number of non nan scores:  364
len of df :  364
(364, 1)
(364, 1)
361
364
364
```



Testing time !!!

In [223]:
```
df_test = pd.read_csv('./test_data/goodreads_data_modified_test.csv');


normed_Star_count_total_test = np.log((df_test['Star_count_total']))
normed_Stars_test = np.log((df_test['Stars']))

df_test['Book score'] = normed_Star_count_total_test + normed_Stars_test


x = np.array(df_test['Book score']).reshape(len(df_test['Book score']), 1)


print np.shape(x)

print np.isfinite(x).sum()
x[np.isneginf(x)] = 0
print np.isfinite(x).sum()

# plt.scatter(x, y,  color='black')
plt.plot(x, logis.predict(x), color='blue', linewidth=3)
plt.xticks(())
plt.yticks(())
plt.show()
```

```
(122, 1)
122
122
```



In [ ]: