Dataset 1: Effects on Learning of teaching experience

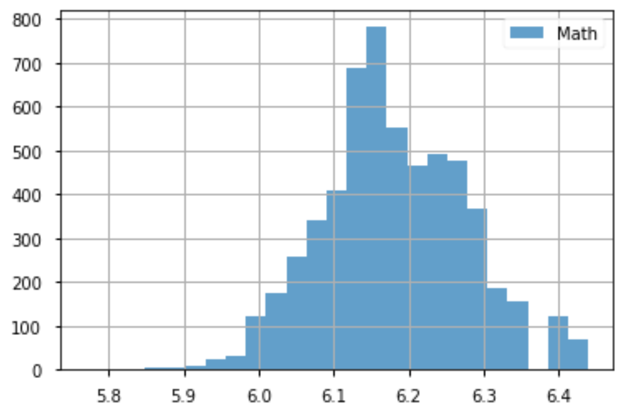
1. Problem Description

The dataset named Star is obtained from the website http://vincentarelbundock.github.io/Rdatasets/datasets.html (http://vincentarelbundock.github.io/Rdatasets/datasets.html). The dataset is a collection of 5748 observations of individual performance in math and reading tests accross 79 different schools. The variables are class sizes, years of teaching experience, gender, qualified for free lunch and race. Below are 5 example instances of the original dataset:

	tmathssk	treadssk	classk	totexpk	sex	freelunk	race	schidkn
2	473	447	small.class	7	girl	no	white	63
3	536	450	small.class	21	girl	no	black	20
5	463	439	regular.with.aide	0	boy	yes	black	19
11	559	448	regular	16	boy	no	white	69
12	489	447	small.class	5	boy	yes	white	79

For the scope of this project work, we focus on choosing only one variable and regress math scores based on that variable. The variable chosen here is years of teaching experience. Comparing the original distribution of math scores vs. log-scale of math scores, log-scale math scores seems to follow normal distribution better. Therefore, we choose to build our models with log-scale math scores.

Data distribution:



2. Model description

In the sections following, we will go through 4 different models:

- Pooled model: all school belongs to the same distribution, log math scores is regressed based on years of teaching experience. Parameters are beta1, beta2 and sigma that are common for all schools.

```
y ~ N(beta1 + beta2*x, sigma)
```

- Separate model: each school has its own separate model. The parameters are alpha (unique for each school), beta and sigma (common for all schools)

```
mu = alpha_school + beta*x
y ~ N(mu, sigma)
```

- Varying slop and intercept model: each school has its own separate model with both alpha and beta unique for each school, sigma is common for all schools. Alpha and beta follows normal distribution with priors:

```
mu_alpha ~ N(0,1)
mu_beta ~ N(0,1)
alpha ~ N(mu_alpha, sigma_alpha)
alpha ~ N(mu_alpha, sigma_alpha)
mu = alpha_school + beta_school*x
```

```
data {
    int<lower=0>  N; //Number of train data
    int<lower=0> M; //Number of test data
    vector[N] x; // Variable - teaching experience
    vector[M] x_test; // Test variable
    vector[N] y; // Labels - log math score
}
parameters {
    vector[2] beta;
    real<lower=0> sigma; //sigma is constrained to be positive
}
model {
    y ~ normal(beta[1] + beta[2]*x, sigma);
}
generated quantities{
    vector[N] log_lik;
    vector[M] y_pred_test;
    for (i in 1:M)
     y_pred_test[i] = normal_rng(beta[1] + beta[2]*x_test[i], sigma);
    for (i in 1:N)
     log_lik[i] = normal_lpdf(y[i]| beta[1] + beta[2]*x[i], sigma);
}
```

2.2. Separate model

Stan code

```
data{
    int<lower=0> N;
    int<lower=1, upper=79> school[N]; #school indicator
    vector[N] x;
    vector[N] y;
}
parameters {
    vector[79] alpha;
    real beta;
    real<lower=0> sigma;
}
transformed parameters {
    vector[N] mu;
    for (i in 1:N)
        mu[i] <- beta* x[i] + alpha[school[i]];
}
model {
    y ~ normal(mu, sigma);
}
generated quantities{
    vector[N] log_lik;
    for (i in 1:N)
     log_lik[i] = normal_lpdf(y[i]| mu[i], sigma);
}
```

2.3. Varying intercept and slope model

Stan code:

```
data {
  int<lower=0> N;
  int<lower=0> J;
  vector[N] y;
  vector[N] x;
  int school[N];
}
parameters {
  real<lower=0> sigma;
  real<lower=0> sigma_a;
  real<lower=0> sigma_b;
  vector[J] alpha;
  vector[J] beta;
  real mu_a;
  real mu_b;
}
transformed parameters {
    vector[N] mu;
    for (i in 1:N)
        mu[i] <- alpha[school[i]] + beta[school[i]]*x[i];
}
model {
  mu_a ~ normal(0, 1);
  mu_b ~ normal(0, 1);
  alpha ~ normal(mu_a, sigma_a);
  beta ~ normal(mu_b, sigma_b);
  y ~ normal(mu, sigma);
}

generated quantities{
    vector[N] log_lik;
    for (i in 1:N)
     log_lik[i] = normal_lpdf(y[i]| mu[i], sigma);
}
```

2.4. Hierarchical model without regressor

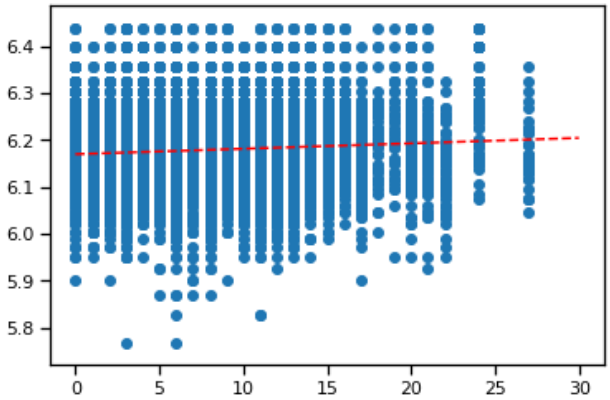
Stan code:

```
data {
    int<lower=1> N;
    int<lower=1> K;
    int<lower=1, upper = K> x[N];
    vector[N] y;
}
parameters {
    real mu0;
    real<lower=0> sigma0;
    vector[K] mu;
    real<lower=0> sigma;
}
model {
    mu ~ normal(mu0, sigma0);
    y~ normal(mu[x], sigma);
}
generated quantities {
    real mupred;
    real ypred;
    vector[N] log_lik;
    mupred = normal_rng(mu0, sigma0);
    ypred = normal_rng(mupred, sigma);
    for (i in 1:N)
        log_lik[i] = normal_lpdf(y[i] | mu[x[i]], sigma);
}
```

3. Convergence and Result

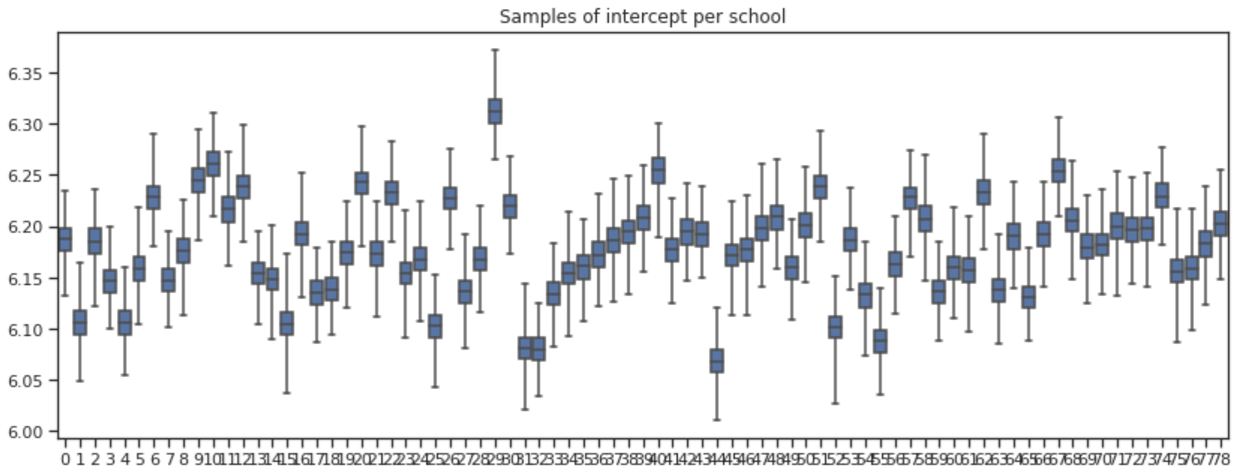
3.1. Pooled model

R-hat for all variables is approximately 1.0, so we can conclude that the model has converged. Checking effective sample size and divergence returns value True, meaning that there is no problem with the model. The graph below demonstrates the fitted line.



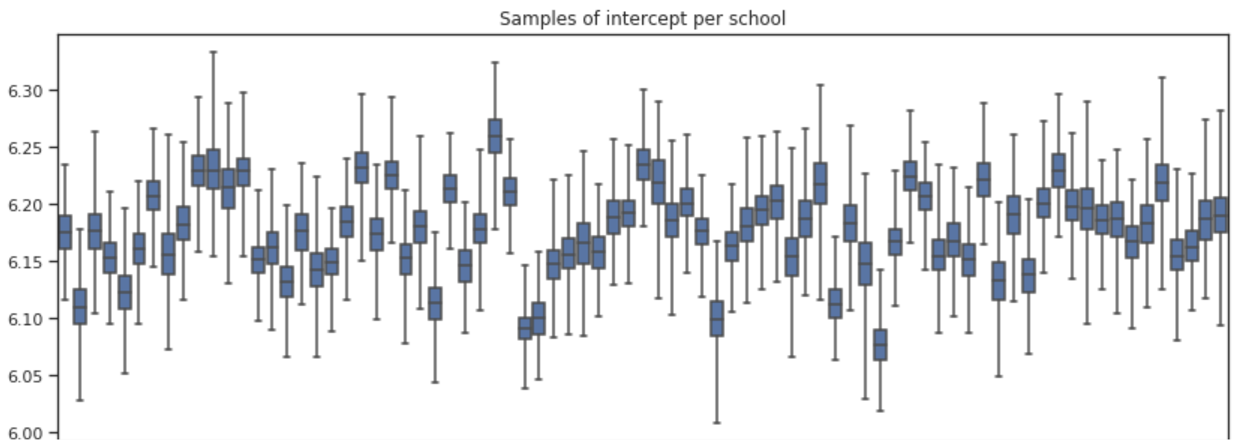
3.2. Separate model

R-hat for all variables is approximately 1.0, so we can conclude that the model has converged. Checking effective sample size and divergence returns value True, meaning that there is no problem with the model. Alpha (intercept) calculated for different schools:



3.3. Varying intercept and slope model

R-hat for all variables is approximately 1.0, so we can conclude that the model has converged. Checking effective sample size and divergence returns value True, meaning that there is no problem with the model. Alpha (intercept) calculated for different schools:



4. Model evaluation with Psis-loo

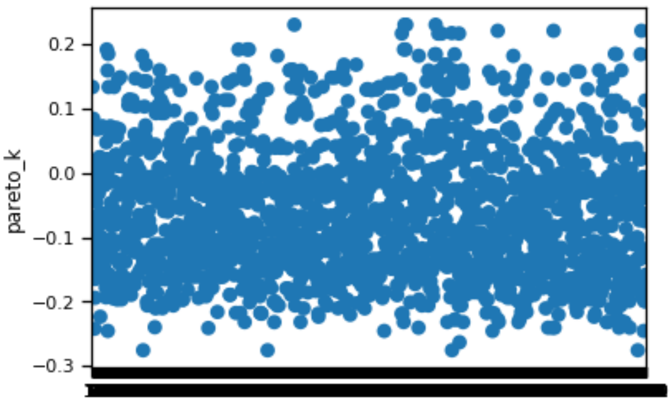
Two methods for model evaluation has been used in this project: psis-loo and MAE.

Psisloo results for each model:

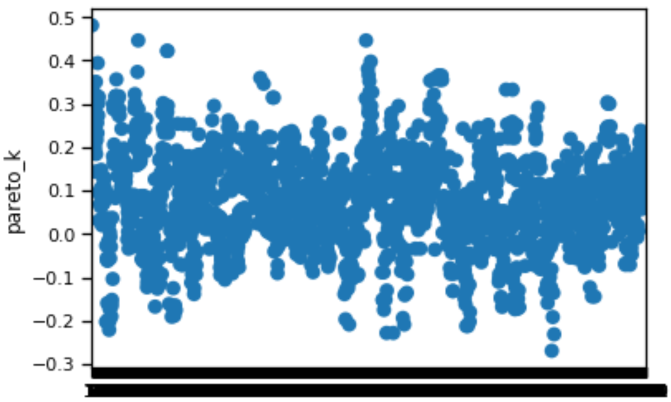
Measurements	Pooled model	Separate model	Varying intercept and slope model	Hierarchical model
psis-loo	1979	2181	2200	2186.7
p_eff	3	79.5	89.2	69
k > 0.5	None	None	None	Some

Scatter plot of ks values for different models:

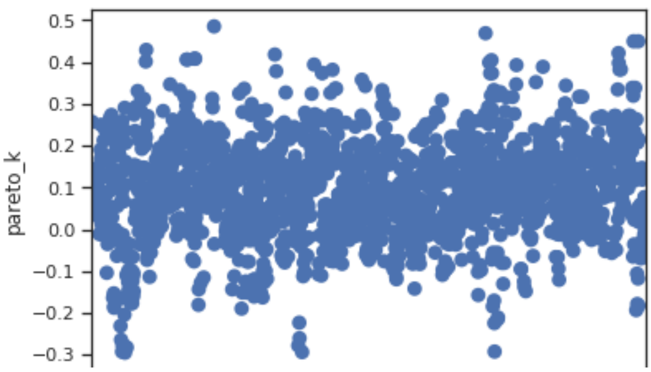
- Pooled model: all k values is below 0.5. We can conclude that the parameter estimations of the model is reliable



- Separate model: all k values is below 0.5. We can conclude that the parameter estimations of the model is reliable.



- Varying intercept and slope model: all k values is below 0.5. We can conclude that the parameter estimations of the model is reliable.



In [1]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns; sns.set_context('notebook')
import stanity
import pystan
```

In [2]:
```
from sklearn.metrics import mean_absolute_error as MAE
from sklearn.metrics import mean_squared_error as MSE
```

In [3]:
```
np.random.seed(100)
import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
star = pd.read_csv('Star.csv', index_col=0, header = 0)
```

In [5]:
```
star.head()
```

Out[5]:

	tmathssk	treadssk	classk	totexpk	sex	freelunk	race	schidkn
2	473	447	small.class	7	girl	no	white	63
3	536	450	small.class	21	girl	no	black	20
5	463	439	regular.with.aide	0	boy	yes	black	19
11	559	448	regular	16	boy	no	white	69
12	489	447	small.class	5	boy	yes	white	79

In [6]:
```
len(np.unique(star.schidkn))
```

Out[6]: 79

In [7]:
```
schools = star.schidkn.unique()
```

In [8]:
```
np.sort(schools)
```

Out[8]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
       52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
       69, 70, 71, 72, 73, 74, 75, 76, 78, 79, 80])
```

In [9]:
```
star.loc[star.schidkn>=78,'schidkn'] -= 1
```

In [10]:
```
schools = star.schidkn.unique()
```

In [11]:
```
df = pd.DataFrame()
```

In [12]:
```
for col in ['tmathssk', 'treadssk', 'totexpk', 'schidkn' ]:
    data = star.loc[:,col]
    df[col] = data
data = star.loc[:,['classk', 'sex', 'freelunk','race']]
df = df.merge(pd.get_dummies(data), left_index=True, right_index=True)
```

In [13]:
```
print(df.columns[[4, 6, 8]])
print(df.head())
```

```
Index(['classk_regular', 'classk_small.class', 'sex_girl'], dtype='object')
    tmathssk  treadssk  totexpk  schidkn  classk_regular  \
2        473       447        7       63               0
3        536       450       21       20               0
5        463       439        0       19               0
11       559       448       16       69               1
12       489       447        5       78               0

    classk_regular.with.aide  classk_small.class  sex_boy  sex_girl  \
2                          0                   1        0         1
3                          0                   1        0         1
5                          1                   0        1         0
11                         0                   0        1         0
12                         0                   1        1         0

    freelunk_no  freelunk_yes  race_black  race_other  race_white
2             1             0           0           0           1
3             1             0           1           0           0
5             0             1           1           0           0
11            1             0           0           0           1
12            0             1           0           0           1
```

In [14]:
```
df.drop(df.columns[[5, 7, 9, 12]], axis=1, inplace=True)
```

In [15]:
```
df.columns = ['math', 'reading', 'year_teaching', 'school','reg_class', 'sml_cl
ass', 'is_girl', 'free_lunch', 'black','white']
```

In [16]:
```
df.head()
```

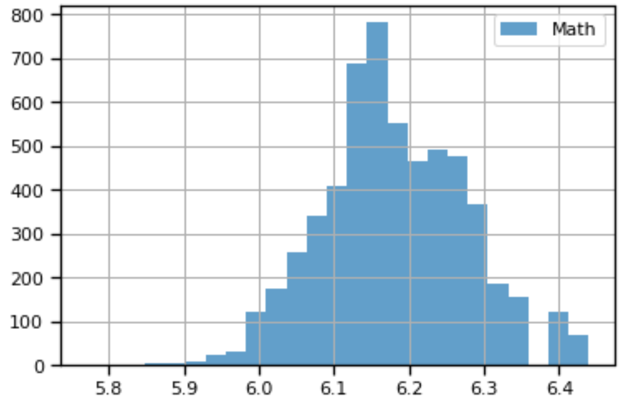
Out[16]:

	math	reading	year_teaching	school	reg_class	sml_class	is_girl	free_lunch	black	white
2	473	447	7	63	0	1	1	0	0	1
3	536	450	21	20	0	1	1	0	1	0
5	463	439	0	19	0	0	0	1	1	0
11	559	448	16	69	1	0	0	0	0	1
12	489	447	5	78	0	1	0	1	0	1

In [17]:
```
df.corr()['math'].sort_values()
```

Out[17]:
```
free_lunch      -0.243111
black           -0.174493
reg_class       -0.036504
school           0.045439
sml_class        0.080078
is_girl          0.081041
year_teaching    0.096687
white            0.174968
reading          0.713549
math             1.000000
Name: math, dtype: float64
```

```
In [18]: df.math.apply(lambda x:np.log(x)).hist(bins=25, alpha = 0.7, label = "Math")
         plt.legend()
         plt.show()
```



Train/Test Split

```
In [19]: X = df.drop('math', axis=1)
         Y = np.log(df.math)
```

```
In [20]: indices = range(df.shape[0])
         i_schools = []
         len_ischools = []
```

```
In [21]: for s in schools:
             i_school = np.where(df.school == s)[0]
             len_ischools.append(len(i_school))
             i_schools.append(i_school)
```

```
In [22]: min_len = min(len_ischools)
         train_size = round(min_len*0.8)
```

```
In [23]: train_idx = []
         test_idx = []
```

```
In [24]: for s in i_schools:
             np.random.shuffle(s)
             train_idx = np.concatenate((train_idx, s[:train_size]), axis=None)
             test_idx = np.concatenate((test_idx, s[train_size:]), axis=None)
```

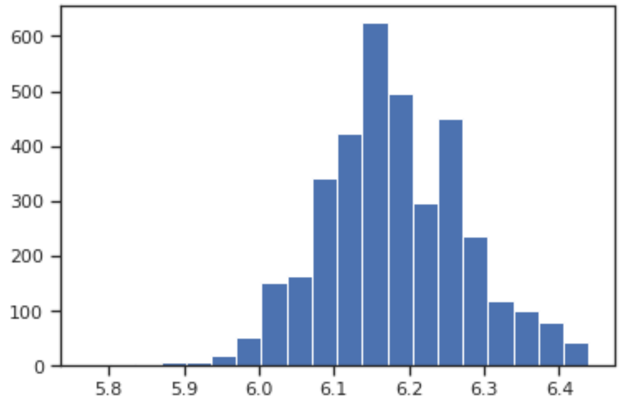
```
In [25]: len(train_idx)

Out[25]: 2133
```

```
In [26]: len(test_idx)

Out[26]: 3615
```

```
In [27]: X_train = X.iloc[train_idx,:]
         X_test = X.iloc[test_idx,:].reset_index()
         Y_train = Y.iloc[train_idx]
         Y_test = Y.iloc[test_idx]
```

In [96]:
```
plt.hist(Y_test, bins = 20)
```

Out[96]:
```
(array([  1.,   3.,   0.,   8.,   6.,  19.,  51., 151., 165., 342., 423.,
        625., 495., 296., 452., 236., 119., 100.,  79.,  44.]),
 array([5.768321  , 5.80187246, 5.83542393, 5.8689754 , 5.90252687,
        5.93607834, 5.96962981, 6.00318128, 6.03673275, 6.07028421,
        6.10383568, 6.13738715, 6.17093862, 6.20449009, 6.23804156,
        6.27159303, 6.3051445 , 6.33869596, 6.37224743, 6.4057989 ,
        6.43935037]),
 <a list of 20 Patch objects>)
```



Pooled model

In [30]:
```
pooled_code = """
data {
    int<lower=0>  N;
    int<lower=0> M;
    vector[N] x;
    vector[M] x_test;
    vector[N] y;
}
parameters {
    vector[2] beta;
    real<lower=0> sigma;
}
model {
    y ~ normal(beta[1] + beta[2]*x, sigma);
}
generated quantities{
    vector[N] log_lik;
    vector[M] y_pred_test;
    for (i in 1:M)
     y_pred_test[i] = normal_rng(beta[1] + beta[2]*x_test[i], sigma);
    for (i in 1:N)
     log_lik[i] = normal_lpdf(y[i]| beta[1] + beta[2]*x[i], sigma);
}

"""
```

In [31]:
```
pool_data = {
    'N': Y_train.shape[0],
    'M': Y_test.shape[0],
    'x': X_train.loc[:,'year_teaching'],
    'x_test': X_test.loc[:,'year_teaching'],
    'y': Y_train,
}
```

In [32]:
```
pooled_fit = pystan.stan(model_code=pooled_code, data = pool_data, iter = 1000,
chains = 2)
```

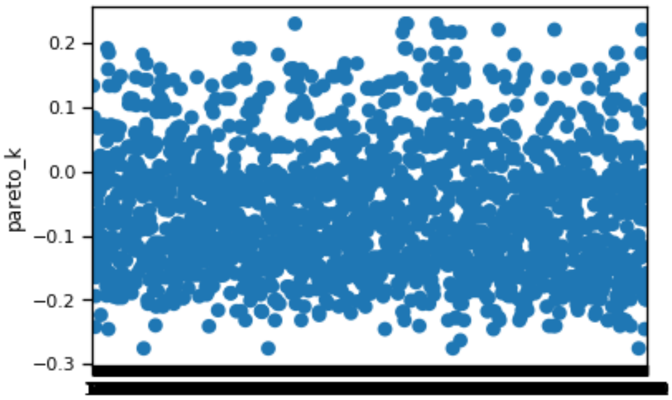
```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_174aac4ec68a4e0adc2676
445f7b9556 NOW.
```

In [33]:
```
print("Rhat check : ",pystan.diagnostics.check_rhat(pooled_fit))
print("N_eff check : ",pystan.diagnostics.check_n_eff(pooled_fit))
print("Divergence check: ", pystan.diagnostics.check_div(pooled_fit))
```

```
Rhat check :  True
N_eff check :  True
Divergence check:  True
```

In [34]:
```
pooled_sample = pooled_fit.extract(permuted=True)

loglik = (pooled_sample["log_lik"])
psisloo = stanity.psisloo(loglik)
print("psisloo: ",psisloo.elpd)
psisloo.plot()
lppd_pooled = 0
for i in range (Y_train.shape[0]):
    lppd_pooled = lppd_pooled + np.log(np.mean(np.exp(loglik[:,i])))

peff_pooled = np.sum(lppd_pooled) - psisloo.elpd
print("p_eff: ", peff_pooled)
```

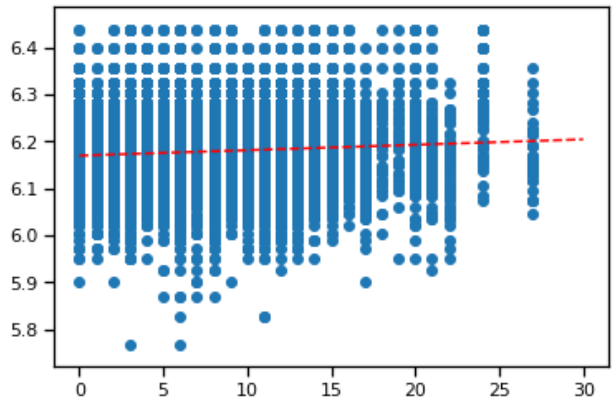
```
psisloo:  1979.3953656872636
p_eff:  3.0732185158378797
```



In [35]:
```
psisloo.print_summary()
```

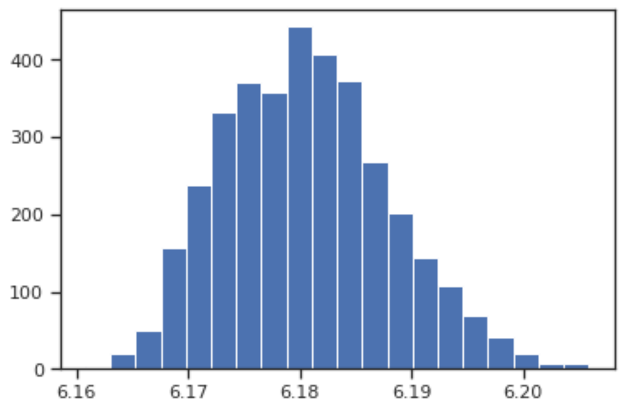
Out[35]:
```
greater than 0.5    0.0
greater than 1      0.0
dtype: float64
```

In [36]:
```
b0, m0 = pooled_sample['beta'].T.mean(1)
plt.scatter(df.year_teaching, np.log(df.math))
xvals = np.linspace(0,30)
plt.plot(xvals, m0*xvals+b0, 'r--')
plt.show()
```



In [93]:
```
#Comparing y_pred with y_test
y_pred = np.mean(pooled_sample["y_pred_test"], axis=0)
plt.hist(y_pred, bins = 20)
print(round(MAE(Y_test, y_pred),3))
```

```
0.077
```



In [38]:
```
round(MSE(Y_test, y_pred),3)
```

Out[38]: 0.01

Separate model

In [39]:
```
separate_code = """
data{
    int<lower=0> N;
    int<lower=1, upper=79> school[N]; #school indicator
    vector[N] x;
    vector[N] y;
}
parameters {
    vector[79] alpha;
    real beta; #only 1 beta, not a vector
    real<lower=0> sigma;
}
transformed parameters {
    vector[N] mu;
    for (i in 1:N)
        mu[i] <- beta* x[i] + alpha[school[i]];
}
model {
    y ~ normal(mu, sigma);
}
generated quantities{
    vector[N] log_lik;
    for (i in 1:N)
     log_lik[i] = normal_lpdf(y[i]| mu[i], sigma);
}
"""
```

In [40]:
```
separate_data = {'N': X_train.shape[0],
                 'school': X_train.school,
                 'x': X_train.loc[:,'year_teaching'],
                 'y': Y_train }
```

In [74]:
```
separate_fit = pystan.stan(model_code = separate_code, data = separate_data, it
er = 1000, chains = 2)
separate_sample = separate_fit.extract(permuted=True)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_3f346b560546d258f09449
1925426d10 NOW.
```

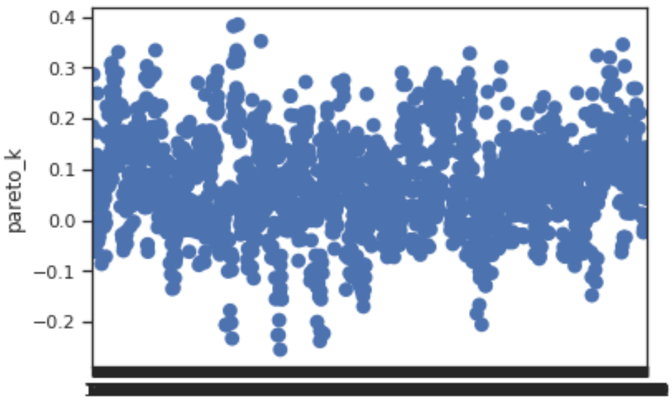
In [75]:
```
print("Rhat check : ",pystan.diagnostics.check_rhat(separate_fit))
print("N_eff check : ",pystan.diagnostics.check_n_eff(separate_fit))
print("Divergence check: ", pystan.diagnostics.check_div(separate_fit))
```

```
Rhat check :  True
N_eff check :  True
Divergence check:  True
```

In [76]:
```
alpha = np.mean(separate_fit['alpha'], axis=0)
beta = np.mean(separate_fit['beta'])
```

In [77]:
```
loglik = (separate_sample["log_lik"])
psisloo = stanity.psisloo(loglik)
print("psisloo: ",psisloo.elpd)
psisloo.plot()
lppd_pooled = 0
for i in range (Y_train.shape[0]):
    lppd_pooled = lppd_pooled + np.log(np.mean(np.exp(loglik[:,i])))

peff_pooled = np.sum(lppd_pooled) - psisloo.elpd
print("p_eff: ", peff_pooled)
```

```
psisloo:  2180.2798747221113
p_eff:  80.5566151392045
```

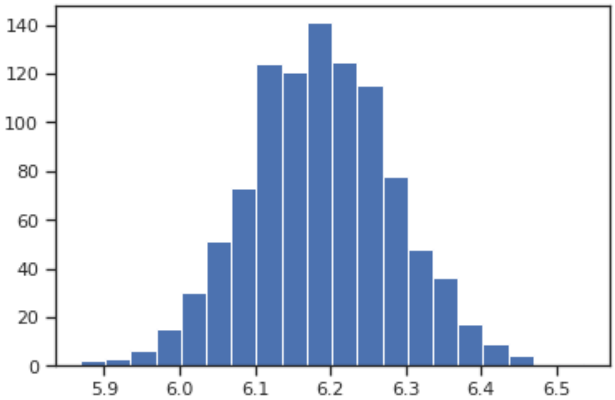


In [78]:
```
psisloo.print_summary()
```

Out[78]:
```
greater than 0.5    0.0
greater than 1      0.0
dtype: float64
```

In [79]:
```
Y_pred = []
for i in range(X_test.shape[0]):
    school = X_test.loc[i, 'school']
    Y_pred.append(alpha[school-1] + beta*X_test.loc[i, 'year_teaching'])

plt.hist(Y_pred, bins = 20)
```

Out[79]:
```
(array([  2.,   3.,   6.,  15.,  30.,  51.,  73., 124., 121., 141., 125.,
        115.,  78.,  48.,  36.,  17.,   9.,   4.,   1.,   1.]),
 array([5.86804705, 5.90145112, 5.93485518, 5.96825925, 6.00166332,
        6.03506738, 6.06847145, 6.10187552, 6.13527958, 6.16868365,
        6.20208772, 6.23549178, 6.26889585, 6.30229992, 6.33570398,
        6.36910805, 6.40251212, 6.43591618, 6.46932025, 6.50272432,
        6.53612838]),
 <a list of 20 Patch objects>)
```

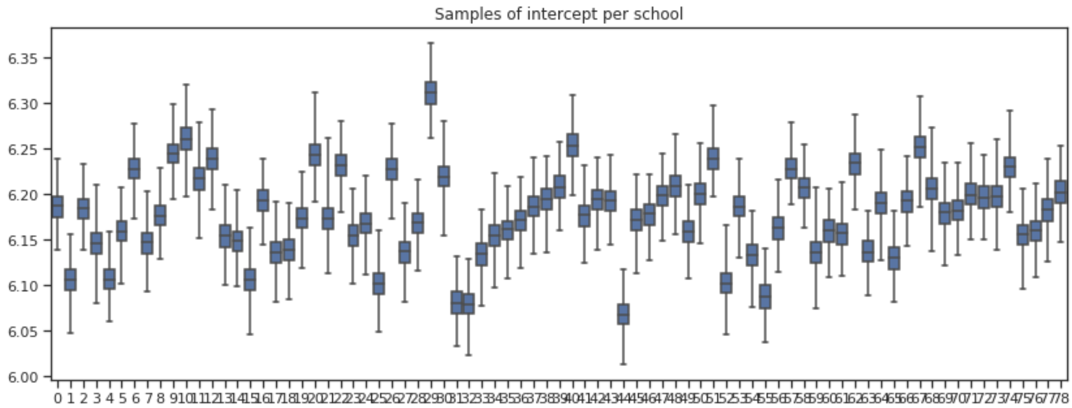


In [80]:
```
print(round(MAE(Y_test, Y_pred),3))
round(MSE(Y_test, Y_pred),3)
```

```
0.07
```

Out[80]:
```
0.008
```

In [98]:
```
import seaborn as sns
sns.set(style="ticks")

a_sample = pd.DataFrame(separate_fit['alpha'])
plt.figure(figsize=(14, 5))
sns.boxplot(data=a_sample, whis=np.inf, color="b")
plt.title("Samples of intercept per school")
plt.show()
```



Varying intercept and slope model

In [48]:

```
varying_intercept_slope = """
data {
  int<lower=0> N;
  int<lower=0> J;
  vector[N] y;
  vector[N] x;
  int school[N];
}
parameters {
  real<lower=0> sigma;
  real<lower=0> sigma_a;
  real<lower=0> sigma_b;
  vector[J] alpha;
  vector[J] beta;
  real mu_a;
  real mu_b;
}
transformed parameters {
    vector[N] mu;
    for (i in 1:N)
        mu[i] <- alpha[school[i]] + beta[school[i]]*x[i];
}
model {
  mu_a ~ normal(0, 1);
  mu_b ~ normal(0, 1);
  alpha ~ normal(mu_a, sigma_a);
  beta ~ normal(mu_b, sigma_b);
  y ~ normal(mu, sigma);
}

generated quantities{
    vector[N] log_lik;
    for (i in 1:N)
     log_lik[i] = normal_lpdf(y[i]| mu[i], sigma);
}

"""
```

In [49]:

```
varying_intercept_slope_data = {'N': X_train.shape[0],
                           'J': 79,
                           'school': X_train.school,
                           'x': X_train.loc[:,'year_teaching'],
                           'y': Y_train}

varying_intercept_slope_fit = pystan.stan(model_code=varying_intercept_slope,
                                          data=varying_intercept_slope_data,
                                          iter=1000, chains=2)
varying_intercept_slope_sample = varying_intercept_slope_fit.extract(permuted=True)
```

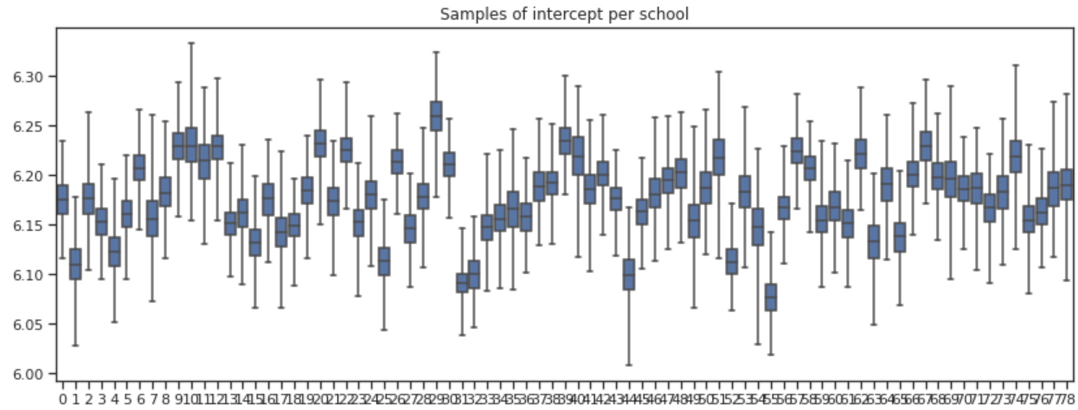
```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_fc996b18191bb3d6f59ec4
3aa9146fea NOW.
```

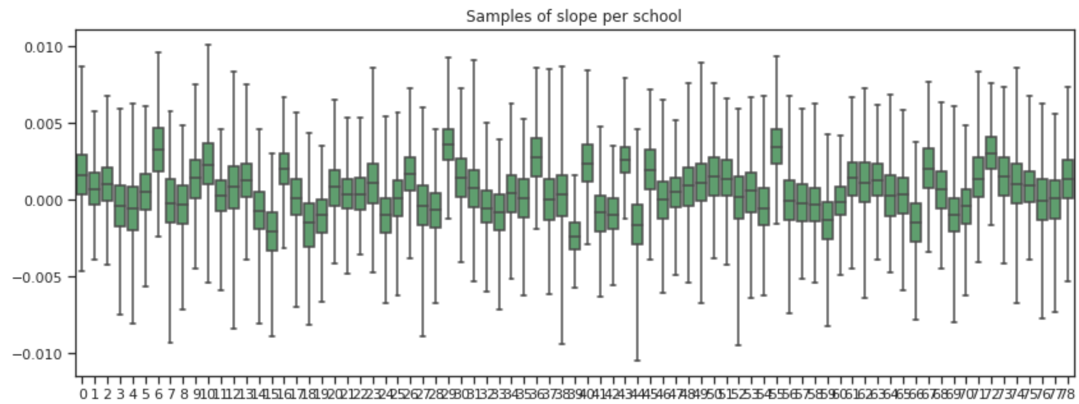
In [50]:

```
print("Rhat check : ",pystan.diagnostics.check_rhat(varying_intercept_slope_fit))
print("N_eff check : ",pystan.diagnostics.check_n_eff(varying_intercept_slope_fit))
print("Divergence check: ", pystan.diagnostics.check_div(varying_intercept_slope_fit))
```

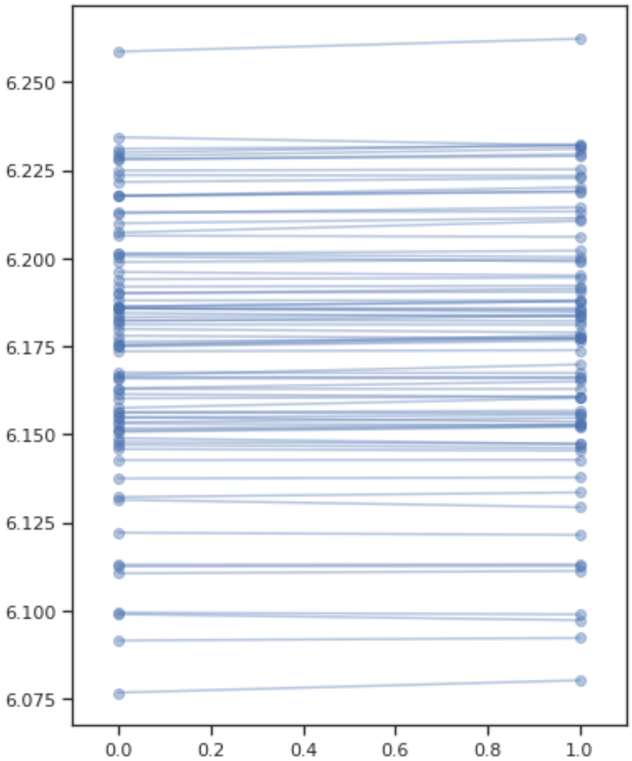
```
Rhat check :  True
N_eff check :  True
Divergence check:  True
```

In [51]:
```
sns.set(style="ticks")

a_sample = pd.DataFrame(varying_intercept_slope_sample['alpha'])
plt.figure(figsize=(14, 5))
sns.boxplot(data=a_sample, whis=np.inf, color="b")
plt.title("Samples of intercept per school")
plt.show()

b_sample = pd.DataFrame(varying_intercept_slope_sample['beta'])
plt.figure(figsize=(14, 5))
sns.boxplot(data=b_sample, whis=np.inf, color = 'g')
plt.title("Samples of slope per school")
plt.show()
```



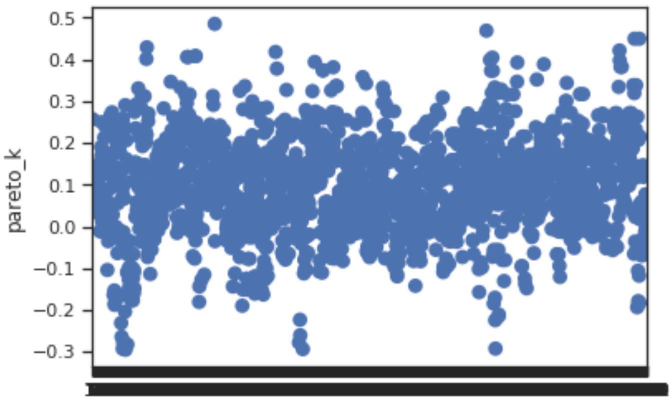


In [52]:
```
xvals = np.arange(2)
b = varying_intercept_slope_fit['alpha'].mean(axis=0)
m = varying_intercept_slope_fit['beta'].mean(axis=0)
plt.figure(figsize=(6,8))
for bi,mi in zip(b,m):
    plt.plot(xvals, mi*xvals + bi, 'bo-', alpha=0.4)
plt.xlim(-0.1, 1.1);
```



In [53]:
```
loglik = (varying_intercept_slope_sample["log_lik"])
psisloo = stanity.psisloo(loglik)
print("psisloo: ",psisloo.elpd)
psisloo.plot()
lppd_pooled = 0
for i in range (Y_train.shape[0]):
    lppd_pooled = lppd_pooled + np.log(np.mean(np.exp(loglik[:,i])))

peff_pooled = np.sum(lppd_pooled) - psisloo.elpd
print("p_eff: ", peff_pooled)
```

```
psisloo:  2200.2774648380127
p_eff:  89.20003702040731
```



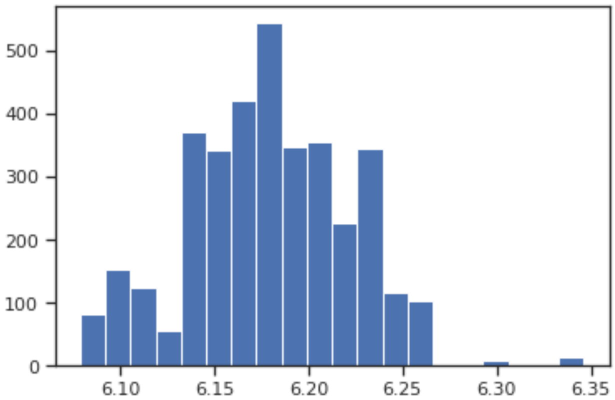
In [54]:
```
psisloo.print_summary()
```

Out[54]:
```
greater than 0.5    0.0
greater than 1      0.0
dtype: float64
```

In [89]:
```
y_pred = []
for i in range(len(Y_test)):
    schoolindx = X_test.school[i] - 1
    y_pred.append(b[schoolindx] + m[schoolindx]*X_test.year_teaching[i])
print(round(MAE(Y_test, y_pred),3))
round(MSE(Y_test, y_pred),3)
plt.hist(y_pred, bins = 20)
```

```
0.07
```

Out[89]:
```
(array([ 82., 154., 125.,  57., 370., 341., 422., 544., 347., 356., 228.,
        346., 117., 103.,   0.,   0.,   9.,   0.,   0.,  14.]),
 array([6.07863881, 6.09202034, 6.10540186, 6.11878338, 6.1321649 ,
        6.14554643, 6.15892795, 6.17230947, 6.185691  , 6.19907252,
        6.21245404, 6.22583556, 6.23921709, 6.25259861, 6.26598013,
        6.27936165, 6.29274318, 6.3061247 , 6.31950622, 6.33288774,
        6.34626927]),
 <a list of 20 Patch objects>)
```



Hierarchical model with no regression variable

Since the hierarchical model is not differentiated from the separated model in predicting a group that is already present in the training data, we estimate this model's effectiveness by using one school as the predicting data and the other schools as the data feeding to the model.

First, let's look at the code.

In [58]:
```
hierarchical_code = """
data {
    int<lower=1> N;
    int<lower=1> K;
    int<lower=1, upper = K> x[N];
    vector[N] y;
}
parameters {
    real mu0;
    real<lower=0> sigma0;
    vector[K] mu;
    real<lower=0> sigma;
}
model {
    mu ~ normal(mu0, sigma0);
    y~ normal(mu[x], sigma);

}
generated quantities {
    real mupred;
    real ypred;
    vector[K] ypred_exist;
    vector[N] log_lik;
    mupred = normal_rng(mu0, sigma0);
    ypred = normal_rng(mupred, sigma);
    for (i in 1:K)
        ypred_exist[i] = normal_rng(mu[i], sigma);
    for (i in 1:N)
        log_lik[i] = normal_lpdf(y[i] | mu[x[i]], sigma);
}
"""
```

In [59]:
```
data_hierarchical = {'N': X_train.shape[0],
                    'K': 79,
                    'x': X_train.school,
                    'y': Y_train}

fit_hierarchical = pystan.stan(model_code=hierarchical_code,
                                           data=data_hierarchical,
                                           iter=1000, chains=2)
hierarchical_sample = fit_hierarchical.extract(permuted=True)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_77c02fecbda80880398a7a
45dcc6fbda NOW.
```

In [60]:
```
print("Rhat check : ",pystan.diagnostics.check_rhat(fit_hierarchical))
print("N_eff check : ",pystan.diagnostics.check_n_eff(fit_hierarchical))
print("Divergence check: ", pystan.diagnostics.check_div(fit_hierarchical))
```

```
Rhat check :  True
N_eff check :  True
Divergence check:  True
```

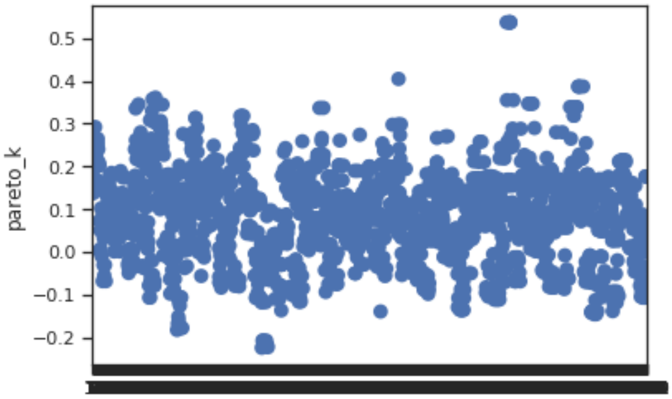
In [61]:

```
loglik = (fit_hierarchical["log_lik"])
psisloo = stanity.psisloo(loglik)
print("psisloo: ",psisloo.elpd)
psisloo.plot()
lppd_pooled = 0
for i in range (Y_train.shape[0]):
    lppd_pooled = lppd_pooled + np.log(np.mean(np.exp(loglik[:,i])))

peff_pooled = np.sum(lppd_pooled) - psisloo.elpd
print("p_eff: ", peff_pooled)
psisloo.print_summary()
```

```
psisloo:  2186.721596639876
p_eff:  68.96740901958947
```

Out[61]:

```
greater than 0.5    0.002344
greater than 1      0.000000
dtype: float64
```



In [62]:

```
mean_error = []
sigma_error = []

mu_pred = np.mean(fit_hierarchical['mupred'])
sigma_pred = np.mean(fit_hierarchical['sigma'])
mu_ytest = np.mean(Y_test)
sigma_ytest = np.std(Y_test)
mean_error.append(mu_ytest - mu_pred)
sigma_error.append(sigma_ytest - sigma_pred)
```

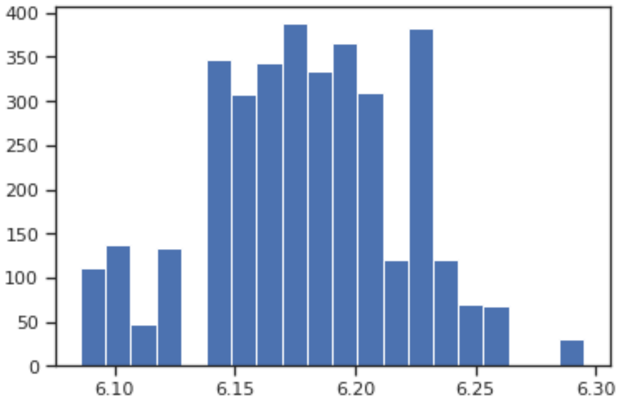
In [68]:

```
Y_pred_test = []

for j in range(len(Y_test)):
    Y_pred_test.append(np.mean(fit_hierarchical["ypred_exist"][:,X_test.school[
j]-1]))
```

In [95]:
```
print(round(MAE(Y_test, Y_pred_test),3))
round(MSE(Y_test, Y_pred_test),3)
plt.hist(Y_pred_test, bins = 20)
```

```
0.07
```

Out[95]:
```
(array([112., 137.,  48., 134.,   0., 347., 308., 343., 388., 334., 365.,
        309., 120., 382., 120.,  69.,  68.,   0.,   0.,  31.]),
 array([6.08569127, 6.09616381, 6.10663635, 6.1171089 , 6.12758144,
        6.13805398, 6.14852653, 6.15899907, 6.16947161, 6.17994415,
        6.1904167 , 6.20088924, 6.21136178, 6.22183433, 6.23230687,
        6.24277941, 6.25325195, 6.2637245 , 6.27419704, 6.28466958,
        6.29514213]),
 <a list of 20 Patch objects>)
```



Dataset 2: New York City Public Schools

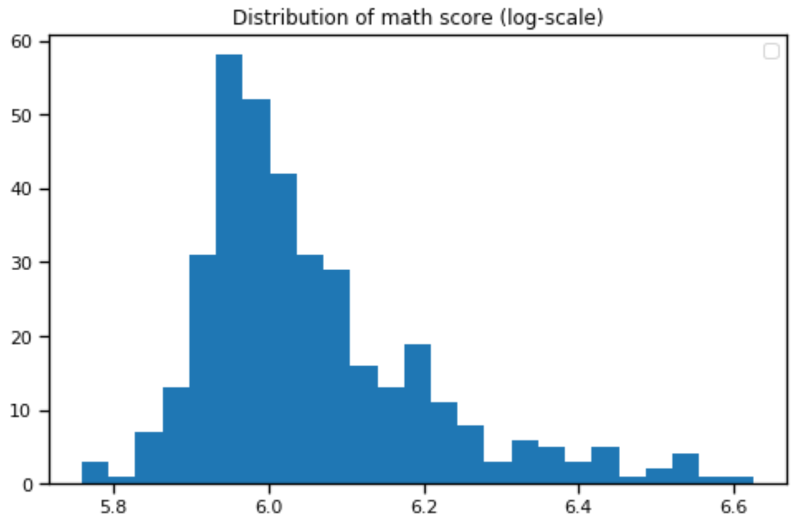
1. Problem Description

The dataset named Score is obtained from the website https://www.kaggle.com/nycopendata/high-schools (https://www.kaggle.com/nycopendata/high-schools). The dataset is a collection of 435 observations of the average score in SAT math, reading and writing tests from all high schools of 5 different boroughs in New York City. Below are the variables of the original dataset:

```
Index: 435 entries, 02M260 to 27Q323
Data columns (total 21 columns):
School Name                    435 non-null object
Borough                        435 non-null object
Building Code                  435 non-null object
Street Address                 435 non-null object
City                           435 non-null object
State                          435 non-null object
Zip Code                       435 non-null int64
Latitude                       435 non-null float64
Longitude                      435 non-null float64
Phone Number                   435 non-null object
Start Time                     431 non-null object
End Time                       431 non-null object
Student Enrollment             428 non-null float64
Percent White                  428 non-null object
Percent Black                  428 non-null object
Percent Hispanic               428 non-null object
Percent Asian                  428 non-null object
Average Score (SAT Math)       375 non-null float64
Average Score (SAT Reading)    375 non-null float64
Average Score (SAT Writing)    375 non-null float64
Percent Tested                 386 non-null object
```

For the scope of this project work, we focus on choosing only one variable and regress SAT average math scores based on that variable. The variable chosen here is the student enrollment. Comparing the original distribution of math scores vs. log-scale of math scores, log-scale math scores seems to follow normal distribution better. Therefore, we choose to build our models with log-scale math scores.

Data distribution:



2. Model description

In the sections following, we will go through 4 different models:

- Pooled model: all borough belongs to the same distribution, log math scores is regressed based on the number of student enrollment. Parameters are beta1, beta2 and sigma that are common for all boroughs.

```
y ~ N(beta1 + beta2*x, sigma)
```

- Separate model: each borough has its own separate model. The parameters are alpha (unique for each borough), beta and sigma (common for all boroughs)

```
mu = alpha_borough + beta*x
```

```
data {
    int<lower=0>  N;
    int<lower=0> M;
    vector[N] x;
    vector[M] x_test;
    vector[N] y;
}
parameters {
    vector[2] beta;
    real<lower=0> sigma;
}
model {
    y ~ normal(beta[1] + beta[2]*x, sigma);
}
generated quantities{
    vector[N] log_lik;
    vector[M] y_pred_test;
    for (i in 1:M)
     y_pred_test[i] = normal_rng(beta[1] + beta[2]*x_test[i], sigma);
    for (i in 1:N)
     log_lik[i] = normal_lpdf(y[i]| beta[1] + beta[2]*x[i], sigma);
}
```

2.2. Separate model

Stan code

```
data{
    int<lower=0> N;
    int<lower=1, upper=5> boroughs[N]; #borough indicator
    vector[N] x;
    vector[N] y;
}
parameters {
    vector[4] alpha;
    real beta; #only 1 beta, not a vector
    real<lower=0> sigma;
}
transformed parameters {
    vector[N] mu;
    for (i in 1:N)
        mu[i] <- beta* x[i] + alpha[boroughs[i]];
}
model {
    y ~ normal(mu, sigma);
}
generated quantities{
    vector[N] log_lik;
    for (i in 1:N)
     log_lik[i] = normal_lpdf(y[i]| mu[i], sigma);
}
```

2.3. Varying intercept and slope model

Stan code:

```
data {
  int<lower=0> N;
  int<lower=0> J;
  vector[N] y;
  vector[N] x;
  int boroughs[N];
}
parameters {
  real<lower=0> sigma;
  real<lower=0> sigma_a;
  real<lower=0> sigma_b;
  vector[J] alpha;
  vector[J] beta;
  real mu_a;
  real mu_b;
}
transformed parameters {
    vector[N] mu;
    for (i in 1:N)
        mu[i] <- alpha[boroughs[i]] + beta[boroughs[i]]*x[i];
}
model {
  mu_a ~ normal(0, 1);
  mu_b ~ normal(0, 1);
  alpha ~ normal(mu_a, sigma_a);
  beta ~ normal(mu_b, sigma_b);
  y ~ normal(mu, sigma);
}
generated quantities{
    vector[N] log_lik;
    for (i in 1:N)
     log_lik[i] = normal_lpdf(y[i]| mu[i], sigma);
}
```

2.4. Hierarchical model

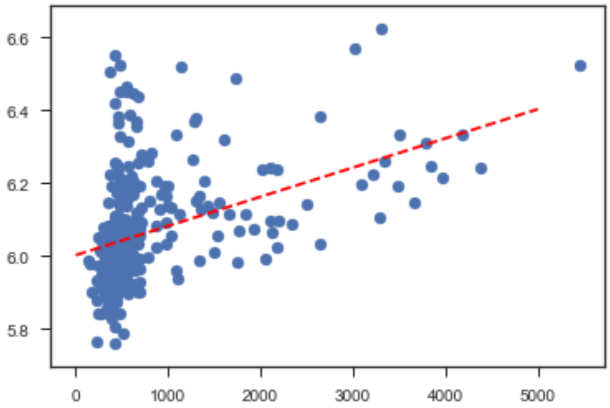
Stan code:

```
data {
    int<lower=1> N;
    int<lower=1> K;
    matrix[N, K] y;
}
parameters {
    real mu0;
    real<lower=0> sigma0;
    vector[K] mu;
    real<lower=0> sigma;
}
model {
    for (j in 1:K){
        mu[j] ~ normal(mu0, sigma0);
        y[:,j] ~ normal(mu[j], sigma);
     }
}
generated quantities {
    real mupred;
    mupred <- normal_rng(mu0, sigma0);
}
```

3. Convergence and Result

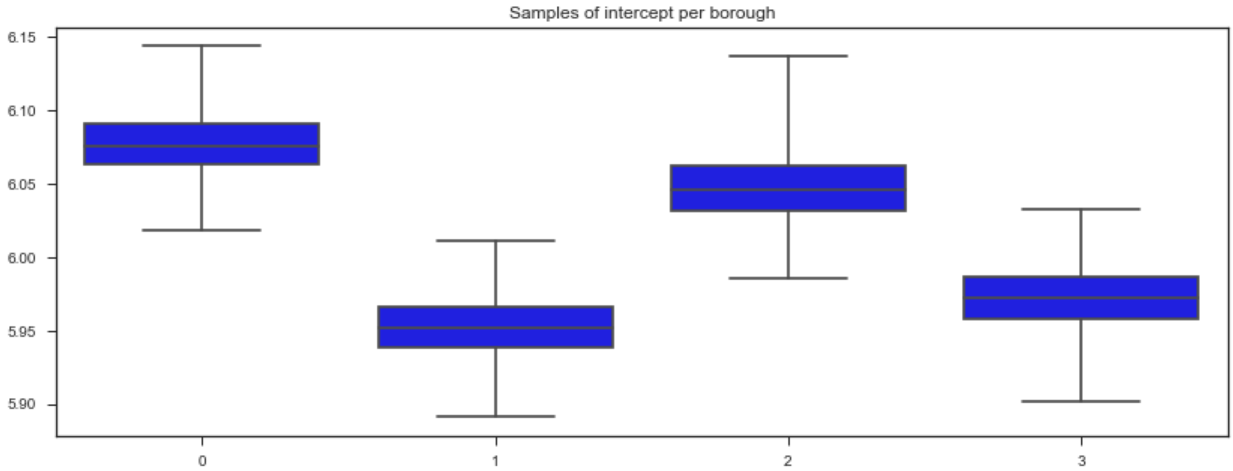
3.1. Pooled model

R-hat for all variables is approximately 1.0, so we can conclude that the model has converged. Checking effective sample size and divergence returns value True, meaning that there is no problem with the model. The graph below demonstrates the fitted line.



3.2. Separate model

R-hat for all variables is approximately 1.0, so we can conclude that the model has converged. Checking effective sample size and divergence returns value True, meaning that there is no problem with the model. Alpha (intercept) calculated for different boroughs:



3.3. Varying intercept and slope model

R-hat for all variables is above 1.1 or below 0.9, so we can conclude that the model has not converged. The model will not be selected for evaluation.

3.4. Hierarchical model

R-hat for all variables is approximately 1.0, so we can conclude that the model has converged. Checking effective sample size returns True, however divergence check returns False, meaning the MCMC estimations maybe biased. The model will not be selected for evaluation.

4. Model evaluation with Psis-loo

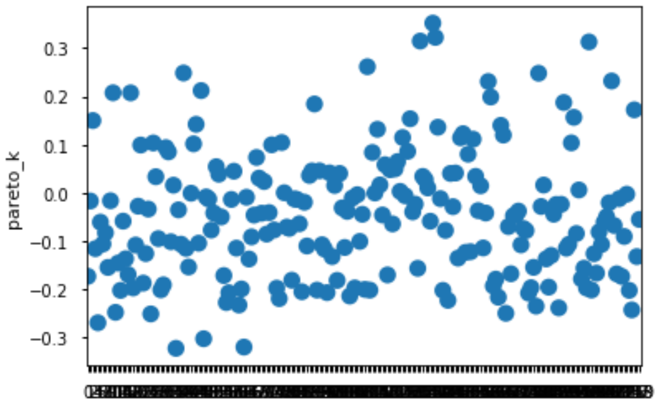
Two methods for model evaluation has been used in this project: psis-loo and MAE.

Psisloo results for each model:

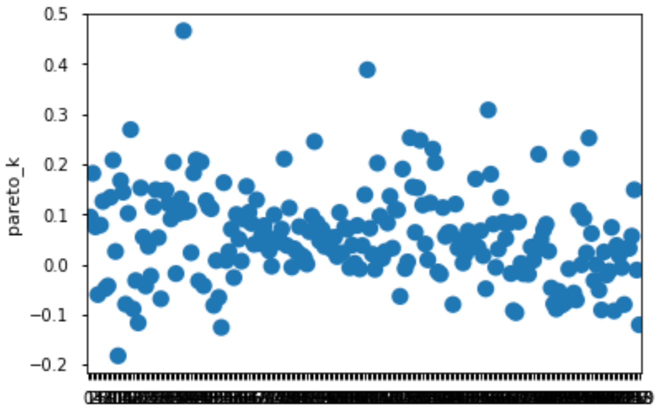
Measurements	Pooled model	Separate model
psis-loo	114.27	125.88
p_eff	4.39	7.23
k > 0.5	None	Some

Scatter plot of ks values for different models:

- Pooled model: all k values is below 0.5. We can conclude that the parameter estimations of the model is reliable.



- Separate model: all k values is below 0.5. We can conclude that the parameter estimations of the model is reliable.



5. Posterior predictive checking

MAE and MSE results for each model:

Measurements	Pooled model	Separate model
MAE	0.094	0.086
MSE	0.014	0.013

Posterior distribution:

- Pooled model

In [41]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns; sns.set_context('notebook')
import stanity
import pystan
import pystan.diagnostics
```

In [42]:
```
from sklearn.metrics import mean_absolute_error as MAE
from sklearn.metrics import mean_squared_error as MSE
```

In [43]:
```
np.random.seed(100)
```

In [44]:
```
score = pd.read_csv('scores.csv', index_col=0, header = 0)
score.head()
```

Out[44]:

	School Name	Borough	Building Code	Street Address	City	State	Zip Code	Latitude	Longitude
School ID									
02M260	Clinton School Writers and Artists	Manhattan	M933	425 West 33rd Street	Manhattan	NY	10001	40.75321	-73.99786
06M211	Inwood Early College for Health and Informatio...	Manhattan	M052	650 Academy Street	Manhattan	NY	10002	40.86605	-73.92486
01M539	New Explorations into Science, Technology and ...	Manhattan	M022	111 Columbia Street	Manhattan	NY	10002	40.71873	-73.97943
02M294	Essex Street Academy	Manhattan	M445	350 Grand Street	Manhattan	NY	10002	40.71687	-73.98953
02M308	Lower Manhattan Arts Academy	Manhattan	M445	350 Grand Street	Manhattan	NY	10002	40.71687	-73.98953

5 rows × 21 columns

In [45]:
```
score.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 435 entries, 02M260 to 27Q323
Data columns (total 21 columns):
School Name                    435 non-null object
Borough                        435 non-null object
Building Code                  435 non-null object
Street Address                 435 non-null object
City                           435 non-null object
State                          435 non-null object
Zip Code                       435 non-null int64
Latitude                       435 non-null float64
Longitude                      435 non-null float64
Phone Number                   435 non-null object
Start Time                     431 non-null object
End Time                       431 non-null object
Student Enrollment             428 non-null float64
Percent White                  428 non-null object
Percent Black                  428 non-null object
Percent Hispanic               428 non-null object
Percent Asian                  428 non-null object
Average Score (SAT Math)       375 non-null float64
Average Score (SAT Reading)    375 non-null float64
Average Score (SAT Writing)    375 non-null float64
Percent Tested                 386 non-null object
dtypes: float64(6), int64(1), object(14)
memory usage: 74.8+ KB
```

In [46]:
```
df = score.loc[:,['School Name', 'Borough', 'Student Enrollment', 'Average Scor
e (SAT Math)']]
df.columns = ['SchoolName', "Borough", "Enrollment", "math"]
df.head()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 435 entries, 02M260 to 27Q323
Data columns (total 4 columns):
SchoolName    435 non-null object
Borough       435 non-null object
Enrollment    428 non-null float64
math          375 non-null float64
dtypes: float64(2), object(2)
memory usage: 17.0+ KB
```

In [47]:
```
df.dropna(inplace=True)
print(df.Borough.value_counts())
```

```
Brooklyn         109
Bronx             98
Manhattan         89
Queens            69
Staten Island     10
Name: Borough, dtype: int64
```

In [48]:
```
st_island = df.index[df.Borough == 'Staten Island']
```

In [49]:
```
df.shape
```

Out[49]: (375, 4)

In [50]:
```
test_df = df.loc[st_island, :]
```

In [51]:
```
df.drop(st_island, inplace=True)
```

In [52]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 365 entries, 01M539 to 27Q323
Data columns (total 4 columns):
SchoolName    365 non-null object
Borough       365 non-null object
Enrollment    365 non-null float64
math          365 non-null float64
dtypes: float64(2), object(2)
memory usage: 14.3+ KB
```

In [53]:
```
df.Borough.unique()
```

Out[53]:
```
array(['Manhattan', 'Bronx', 'Queens', 'Brooklyn'], dtype=object)
```

In [54]:
```
replace_map = {'Borough':{'Manhattan': 1, 'Bronx': 2, 'Queens':3, 'Brooklyn':4}
}
df.replace(replace_map, inplace = True)
print(df.head())
boroughs = df.Borough.unique()
print(boroughs)
```

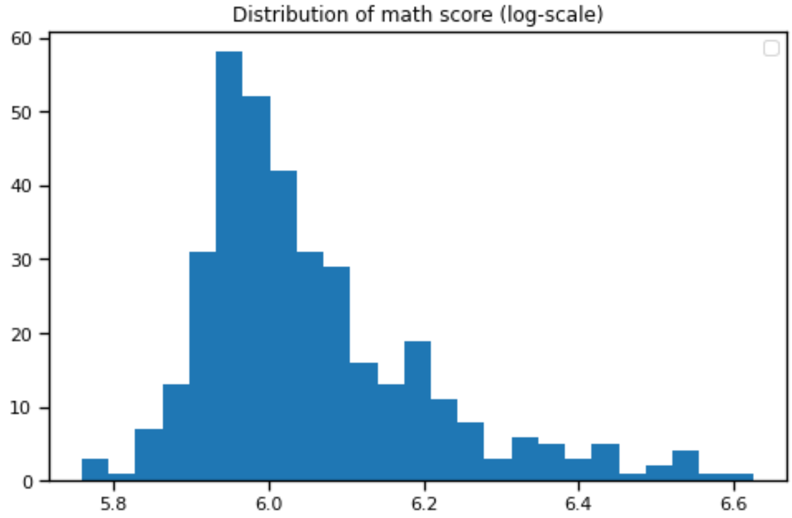
```
                                                    SchoolName  Borough  \
School ID
01M539     New Explorations into Science, Technology and ...        1
02M294                                   Essex Street Academy        1
02M308                           Lower Manhattan Arts Academy        1
02M545       High School for Dual Language and Asian Studies        1
01M292         Henry Street School for International Studies        1

           Enrollment   math
School ID
01M539         1735.0  657.0
02M294          358.0  395.0
02M308          383.0  418.0
02M545          416.0  613.0
01M292          255.0  410.0
[1 2 3 4]
```

In [55]:
```
df.corr()['math'].sort_values()
```

Out[55]:
```
Borough      -0.111108
Enrollment    0.452911
math          1.000000
Name: math, dtype: float64
```

In [56]:
```
plt.figure(figsize = (8,5))
plt.hist(np.log(df.math), bins = 25)
plt.title("Distribution of math score (log-scale)")
plt.legend()
plt.show()
```

```
WARNING:matplotlib.legend:No handles with labels found to put in legend.
```



Train/Test Split

In [57]:
```
X = df.drop('math', axis=1)
Y = np.log(df.math)
```

In [58]:
```
indices = range(df.shape[0])
i_boroughs= []
len_iboroughs = []
```

In [59]:
```
for s in boroughs:
    i_borough = np.where(df.Borough == s)[0]
    len_iboroughs.append(len(i_borough))
    i_boroughs.append(i_borough)
```

In [60]:
```
min_len = min(len_iboroughs)
train_size = round(min_len*0.8)
```

In [61]:
```
train_idx = []
test_idx = []
```

In [62]:
```
for s in i_boroughs:
    np.random.shuffle(s)
    train_idx = np.concatenate((train_idx, s[:train_size]), axis=None)
    test_idx = np.concatenate((test_idx, s[train_size:]), axis=None)
```

In [63]:
```
df.shape[0]
```

Out[63]: 365

In [64]:
```
len(train_idx)
```

Out[64]: 220

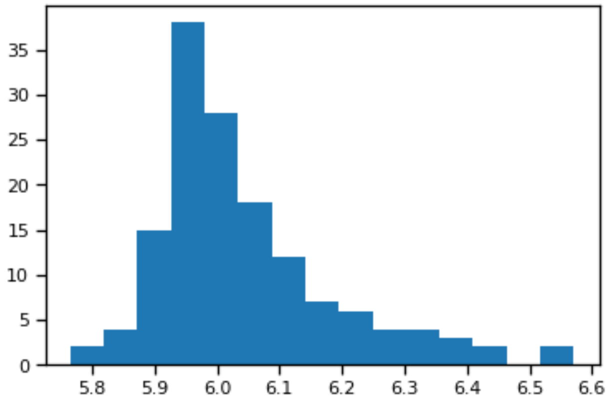
In [65]:
```
len(test_idx)
```

Out[65]:
```
145
```

In [66]:
```
X_train = X.iloc[train_idx,:]
X_test = X.iloc[test_idx,:].reset_index()
Y_train = Y.iloc[train_idx]
Y_test = Y.iloc[test_idx]
```

In [67]:
```
plt.hist(Y_test, bins = 15)
```

Out[67]:
```
(array([ 2.,  4., 15., 38., 28., 18., 12.,  7.,  6.,  4.,  4.,  3.,  2.,
         0.,  2.]),
 array([5.7651911 , 5.81890389, 5.87261668, 5.92632947, 5.98004227,
        6.03375506, 6.08746785, 6.14118064, 6.19489343, 6.24860622,
        6.30231901, 6.3560318 , 6.40974459, 6.46345738, 6.51717017,
        6.57088296]),
 <a list of 15 Patch objects>)
```



Pooled model

In [68]:
```
pooled_code = """
data {
    int<lower=0>  N;
    int<lower=0> M;
    vector[N] x;
    vector[M] x_test;
    vector[N] y;
}
parameters {
    vector[2] beta;
    real<lower=0> sigma;
}
model {
    y ~ normal(beta[1] + beta[2]*x, sigma);
}
generated quantities{
    vector[N] log_lik;
    vector[M] y_pred_test;
    for (i in 1:M)
     y_pred_test[i] = normal_rng(beta[1] + beta[2]*x_test[i], sigma);
    for (i in 1:N)
     log_lik[i] = normal_lpdf(y[i]| beta[1] + beta[2]*x[i], sigma);
}

"""
```

In [69]:
```
pool_data = {
    'N': Y_train.shape[0],
    'M': Y_test.shape[0],
    'x': X_train.loc[:,'Enrollment'],
    'x_test': X_test.loc[:,'Enrollment'],
    'y': Y_train,
}
```

In [70]:
```
pooled_fit = pystan.stan(model_code=pooled_code, data = pool_data, iter = 1000,
chains = 2)
```

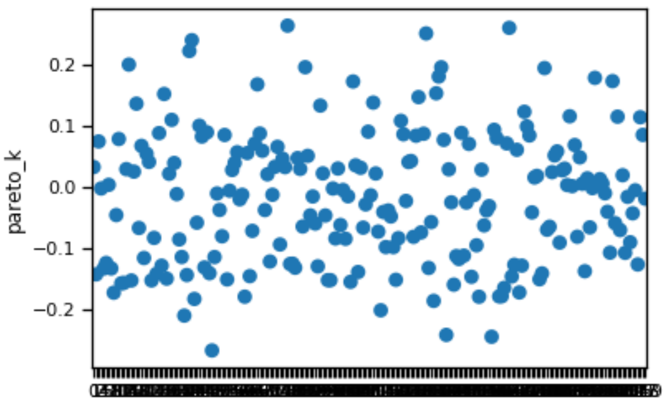
```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_174aac4ec68a4e0adc2676
445f7b9556 NOW.
/opt/conda/lib/python3.6/site-packages/Cython/Compiler/Main.py:367: FutureWarn
ing: Cython directive 'language_level' not set, using 2 for now (Py2). This wi
ll change in a later release! File: /tmp/tmp237wt2hs/stanfit4anon_model_174aac
4ec68a4e0adc2676445f7b9556_8582388925517822565.pyx
  tree = Parsing.p_module(s, pxd, full_module_name)
WARNING:pystan:545 of 1000 iterations saturated the maximum tree depth of 10 (
54.5%)
WARNING:pystan:Run again with max_treedepth larger than 10 to avoid saturation
```

In [71]:
```
print("Rhat check : ",pystan.diagnostics.check_rhat(pooled_fit))
print("N_eff check : ",pystan.diagnostics.check_n_eff(pooled_fit))
print("Divergence check: ", pystan.diagnostics.check_div(pooled_fit))
```

```
Rhat check :  True
N_eff check :  True
Divergence check:  True
```

In [72]:
```
pooled_sample = pooled_fit.extract(permuted=True)

loglik = (pooled_sample["log_lik"])
psisloo = stanity.psisloo(loglik)
print("psisloo: ",psisloo.elpd)
psisloo.plot()
lppd_pooled = 0
for i in range (Y_train.shape[0]):
    lppd_pooled = lppd_pooled + np.log(np.mean(np.exp(loglik[:,i])))

peff_pooled = np.sum(lppd_pooled) - psisloo.elpd
print("p_eff: ", peff_pooled)
```

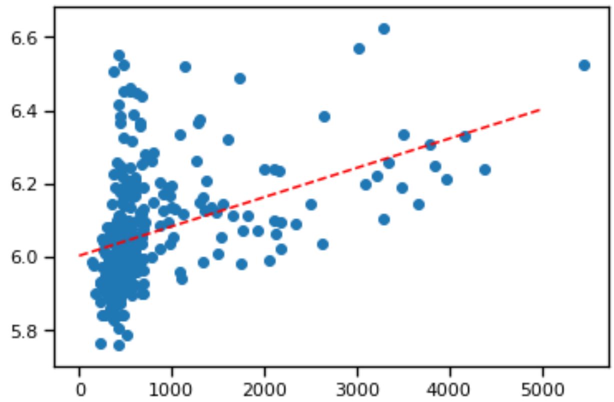
```
psisloo:  114.27120365734639
p_eff:  4.396478635483916
```



In [73]:
```
psisloo.print_summary()
```

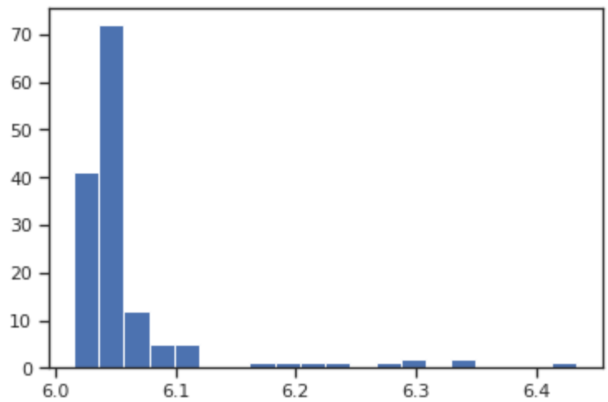
Out[73]:
```
greater than 0.5    0.0
greater than 1      0.0
dtype: float64
```

In [74]:
```
b0, m0 = pooled_sample['beta'].T.mean(1)
plt.scatter(df.Enrollment, np.log(df.math))
xvals = np.linspace(0,5000)
plt.plot(xvals, m0*xvals+b0, 'r--')
plt.show()
```



In [105]:
```
#Comparing y_pred with y_test
y_pred = np.mean(pooled_sample["y_pred_test"], axis=0)
plt.hist(y_pred, bins = 20)
round(MAE(Y_test, y_pred),3)
```

Out[105]: 0.094



In [76]:
```
round(MSE(Y_test, y_pred),3)
```

Out[76]: 0.014

Separate model

In [77]:
```
separate_code = """
data{
    int<lower=0> N;
    int<lower=1, upper=5> boroughs[N]; #borough indicator
    vector[N] x;
    vector[N] y;
}
parameters {
    vector[4] alpha;
    real beta; #only 1 beta, not a vector
    real<lower=0> sigma;
}
transformed parameters {
    vector[N] mu;
    for (i in 1:N)
        mu[i] <- beta* x[i] + alpha[boroughs[i]];
}
model {
    y ~ normal(mu, sigma);
}
generated quantities{
    vector[N] log_lik;
    for (i in 1:N)
     log_lik[i] = normal_lpdf(y[i]| mu[i], sigma);
}
"""
```

In [78]:
```
separate_data = {'N': X_train.shape[0],
                 'boroughs': X_train.Borough,
                 'x': X_train.loc[:,'Enrollment'],
                 'y': Y_train }
```

In [79]:
```
separate_fit = pystan.stan(model_code = separate_code, data = separate_data, it
er = 1000, chains = 2)
separate_sample = separate_fit.extract(permuted=True)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_aae52bba719395c276d180
e52079c523 NOW.
/opt/conda/lib/python3.6/site-packages/Cython/Compiler/Main.py:367: FutureWarn
ing: Cython directive 'language_level' not set, using 2 for now (Py2). This wi
ll change in a later release! File: /tmp/tmpua__yk6i/stanfit4anon_model_aae52b
ba719395c276d180e52079c523_573625073612541637.pyx
  tree = Parsing.p_module(s, pxd, full_module_name)
WARNING:pystan:953 of 1000 iterations saturated the maximum tree depth of 10 (
95.3%)
WARNING:pystan:Run again with max_treedepth larger than 10 to avoid saturation
```

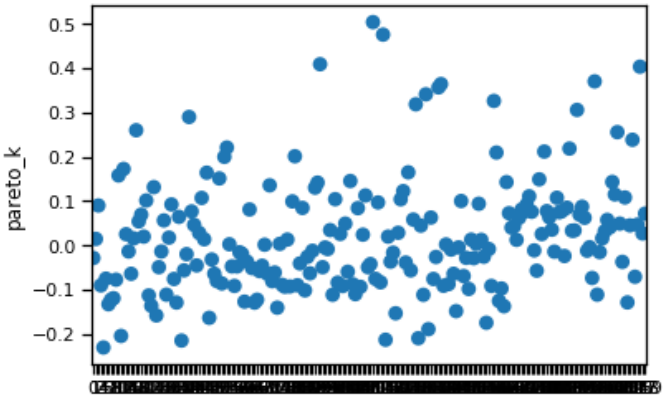
In [80]:
```
print("Rhat check : ",pystan.diagnostics.check_rhat(separate_fit))
print("N_eff check : ",pystan.diagnostics.check_n_eff(separate_fit))
print("Divergence check: ", pystan.diagnostics.check_div(separate_fit))
```

```
Rhat check :  True
N_eff check :  True
Divergence check:  True
```

In [81]:
```
alpha = np.mean(separate_fit['alpha'], axis=0)
beta = np.mean(separate_fit['beta'])
```

In [82]:
```
loglik = (separate_sample["log_lik"])
psisloo = stanity.psisloo(loglik)
print("psisloo: ",psisloo.elpd)
psisloo.plot()
lppd_pooled = 0
for i in range (Y_train.shape[0]):
    lppd_pooled = lppd_pooled + np.log(np.mean(np.exp(loglik[:,i])))

peff_pooled = np.sum(lppd_pooled) - psisloo.elpd
print("p_eff: ", peff_pooled)
```

```
psisloo:  125.87970321486277
p_eff:  7.234032612050086
```

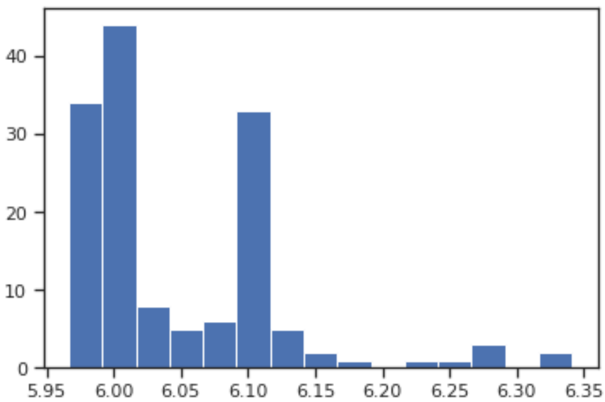


In [83]:
```
psisloo.print_summary()
```

Out[83]:
```
greater than 0.5    0.004545
greater than 1      0.000000
dtype: float64
```

In [98]:
```
Y_pred = []
for i in range(X_test.shape[0]):
    borough = X_test.loc[i, 'Borough']
    Y_pred.append(alpha[borough-1] + beta*X_test.loc[i, 'Enrollment'])

plt.hist(Y_pred, bins = 15)
```

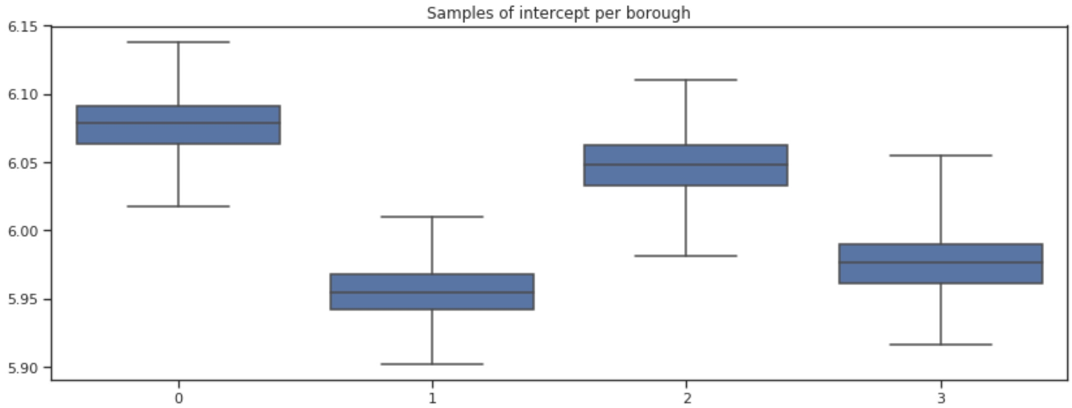
Out[98]:
```
(array([34., 44.,  8.,  5.,  6., 33.,  5.,  2.,  1.,  0.,  1.,  1.,  3.,
         0.,  2.]),
 array([5.96636733, 5.99135656, 6.01634579, 6.04133501, 6.06632424,
        6.09131346, 6.11630269, 6.14129192, 6.16628114, 6.19127037,
        6.21625959, 6.24124882, 6.26623805, 6.29122727, 6.3162165 ,
        6.34120572]),
 <a list of 15 Patch objects>)
```



In [86]:
```
MAE(Y_pred, Y_test)
```
Out[86]: 0.0865400999356269

In [87]:
```
MSE(Y_pred, Y_test)
```
Out[87]: 0.01333292442605992

In [88]:
```
import seaborn as sns
sns.set(style="ticks")

a_sample = pd.DataFrame(separate_fit['alpha'])
plt.figure(figsize=(14, 5))
sns.boxplot(data=a_sample, whis=np.inf, color="b")
plt.title("Samples of intercept per borough")
plt.show()
```



Varying intercept and slope model

In [89]:
```
varying_intercept_slope = """
data {
  int<lower=0> N;
  int<lower=0> J;
  vector[N] y;
  vector[N] x;
  int boroughs[N];
}
parameters {
  real<lower=0> sigma;
  real<lower=0> sigma_a;
  real<lower=0> sigma_b;
  vector[J] alpha;
  vector[J] beta;
  real mu_a;
  real mu_b;
}
transformed parameters {
    vector[N] mu;
    for (i in 1:N)
        mu[i] <- alpha[boroughs[i]] + beta[boroughs[i]]*x[i];
}
model {
  mu_a ~ normal(0, 1);
  mu_b ~ normal(0, 1);

  alpha ~ normal(mu_a, sigma_a);
  beta ~ normal(mu_b, sigma_b);
  y ~ normal(mu, sigma);
}

generated quantities{
    vector[N] log_lik;
    for (i in 1:N)
     log_lik[i] = normal_lpdf(y[i]| mu[i], sigma);
}

"""
```

In [90]:
```
varying_intercept_slope_data = {'N': X_train.shape[0],
                              'J': 4,
                              'boroughs': X_train.Borough,
                              'x': X_train.loc[:,'Enrollment'],
                              'y': Y_train}

varying_intercept_slope_fit = pystan.stan(model_code=varying_intercept_slope,
                                          data=varying_intercept_slope_data,
                                          iter=1000, chains=2)
varying_intercept_slope_sample = varying_intercept_slope_fit.extract(permuted=T
rue)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_989c483a074c9067ebf9bb
4c197bcf99 NOW.
/opt/conda/lib/python3.6/site-packages/Cython/Compiler/Main.py:367: FutureWarn
ing: Cython directive 'language_level' not set, using 2 for now (Py2). This wi
ll change in a later release! File: /tmp/tmpnnmxntyl/stanfit4anon_model_989c48
3a074c9067ebf9bb4c197bcf99_3942053905460856220.pyx
  tree = Parsing.p_module(s, pxd, full_module_name)
WARNING:pystan:Rhat for parameter sigma is 1.2525368313454828!
WARNING:pystan:Rhat for parameter sigma_a is 1.6946946256716302!
WARNING:pystan:Rhat for parameter sigma_b is 1.270313694893328!
WARNING:pystan:Rhat for parameter alpha[2] is 1.4227734713156592!
WARNING:pystan:Rhat for parameter alpha[3] is 1.5751425770229537!
WARNING:pystan:Rhat for parameter alpha[4] is 2.4922765898590375!
WARNING:pystan:Rhat for parameter beta[1] is 1.3085444568367504!
WARNING:pystan:Rhat for parameter beta[2] is 1.148555765076775!
WARNING:pystan:Rhat for parameter beta[3] is 1.5250175242078963!
WARNING:pystan:Rhat for parameter mu_a is 1.9351990593203243!
WARNING:pystan:Rhat for parameter mu[2] is 1.138796604654893!
WARNING:pystan:Rhat for parameter mu[3] is 1.218781425377582!
WARNING:pystan:Rhat for parameter mu[5] is 1.1329334583549957!
WARNING:pystan:Rhat for parameter mu[8] is 1.1012985603437124!
WARNING:pystan:Rhat for parameter mu[9] is 1.272914183469249!
WARNING:pystan:Rhat for parameter mu[10] is 1.113632904069064!
WARNING:pystan:Rhat for parameter mu[11] is 1.187569597575632!
WARNING:pystan:Rhat for parameter mu[12] is 1.1094679324389236!
WARNING:pystan:Rhat for parameter mu[13] is 1.4082115802696265!
WARNING:pystan:Rhat for parameter mu[14] is 1.276055315738049!
WARNING:pystan:Rhat for parameter mu[15] is 1.1087787800158826!
WARNING:pystan:Rhat for parameter mu[16] is 1.2963576993251207!
WARNING:pystan:Rhat for parameter mu[17] is 1.1164370441723843!
WARNING:pystan:Rhat for parameter mu[18] is 1.1314766207266838!
WARNING:pystan:Rhat for parameter mu[19] is 1.391710552807151!
WARNING:pystan:Rhat for parameter mu[20] is 1.2791575842645473!
WARNING:pystan:Rhat for parameter mu[21] is 1.3411289365980965!
WARNING:pystan:Rhat for parameter mu[22] is 1.1178469867881033!
WARNING:pystan:Rhat for parameter mu[25] is 1.415429818165475!
WARNING:pystan:Rhat for parameter mu[26] is 1.1080910996442717!
WARNING:pystan:Rhat for parameter mu[29] is 1.1754993455139824!
WARNING:pystan:Rhat for parameter mu[31] is 1.2929123072211073!
WARNING:pystan:Rhat for parameter mu[32] is 1.1178469867881033!
WARNING:pystan:Rhat for parameter mu[33] is 1.1115440695995307!
WARNING:pystan:Rhat for parameter mu[35] is 1.4101098585718697!
WARNING:pystan:Rhat for parameter mu[37] is 1.1807843590010574!
WARNING:pystan:Rhat for parameter mu[38] is 1.1469397405104818!
WARNING:pystan:Rhat for parameter mu[39] is 1.3656402421553926!
WARNING:pystan:Rhat for parameter mu[40] is 1.2033001436517838!
WARNING:pystan:Rhat for parameter mu[41] is 1.2317612848341253!
WARNING:pystan:Rhat for parameter mu[42] is 1.2238652725019374!
WARNING:pystan:Rhat for parameter mu[43] is 1.405887060970638!
WARNING:pystan:Rhat for parameter mu[44] is 1.268452519818485!
WARNING:pystan:Rhat for parameter mu[46] is 1.1845559142900657!
WARNING:pystan:Rhat for parameter mu[48] is 1.218781425377582!
WARNING:pystan:Rhat for parameter mu[49] is 1.1150323079747213!
WARNING:pystan:Rhat for parameter mu[50] is 1.1566569533578688!
WARNING:pystan:Rhat for parameter mu[51] is 1.414442026682911!
WARNING:pystan:Rhat for parameter mu[53] is 1.404715008587363!
WARNING:pystan:Rhat for parameter mu[54] is 1.4134917173056114!
WARNING:pystan:Rhat for parameter mu[55] is 1.248531745950521!
WARNING:pystan:Rhat for parameter mu[56] is 1.1967699213939647!
WARNING:pystan:Rhat for parameter mu[57] is 1.174879157718029!
WARNING:pystan:Rhat for parameter mu[58] is 1.203573734913662!
WARNING:pystan:Rhat for parameter mu[60] is 1.43051507380884!
WARNING:pystan:Rhat for parameter mu[62] is 1.208163963830866!
WARNING:pystan:Rhat for parameter mu[63] is 1.378089720872561!
WARNING:pystan:Rhat for parameter mu[64] is 1.2280987577661364!
WARNING:pystan:Rhat for parameter mu[65] is 1.3624647358564854!
WARNING:pystan:Rhat for parameter mu[66] is 1.2975351716107302!
WARNING:pystan:Rhat for parameter mu[67] is 1.3180397089494404!
WARNING:pystan:Rhat for parameter mu[69] is 1.3635360590019157!
WARNING:pystan:Rhat for parameter mu[70] is 1.3850330919377793!
```

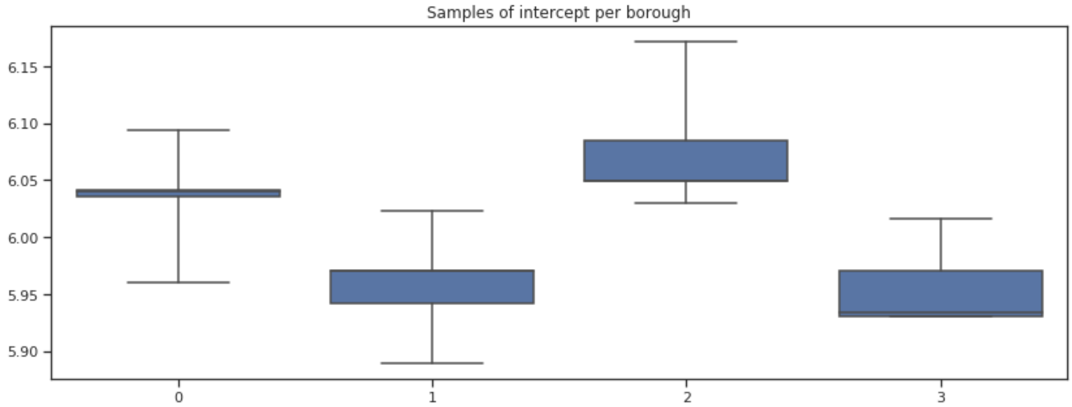
In [91]:

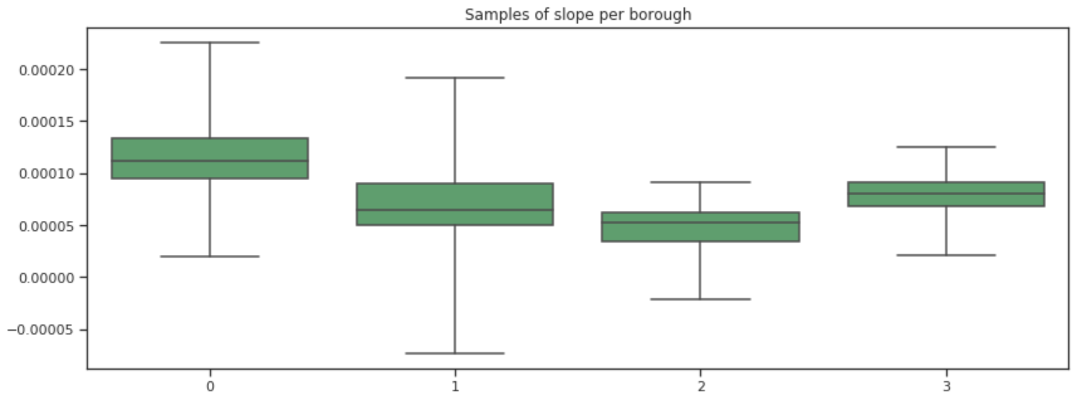
```
print("Rhat check : ",pystan.diagnostics.check_rhat(varying_intercept_slope_fit
))
print("N_eff check : ",pystan.diagnostics.check_n_eff(varying_intercept_slope_f
it))
print("Divergence check: ", pystan.diagnostics.check_div(varying_intercept_slop
e_fit))
```

```
WARNING:pystan:Rhat for parameter sigma is 1.2525368313454828!
WARNING:pystan:Rhat for parameter sigma_a is 1.6946946256716302!
WARNING:pystan:Rhat for parameter sigma_b is 1.270313694893328!
WARNING:pystan:Rhat for parameter alpha[2] is 1.4227734713156592!
WARNING:pystan:Rhat for parameter alpha[3] is 1.5751425770229537!
WARNING:pystan:Rhat for parameter alpha[4] is 2.4922765898590375!
WARNING:pystan:Rhat for parameter beta[1] is 1.3085444568367504!
WARNING:pystan:Rhat for parameter beta[2] is 1.148555765076775!
WARNING:pystan:Rhat for parameter beta[3] is 1.5250175242078963!
WARNING:pystan:Rhat for parameter mu_a is 1.9351990593203243!
WARNING:pystan:Rhat for parameter mu[2] is 1.138796604654893!
WARNING:pystan:Rhat for parameter mu[3] is 1.218781425377582!
WARNING:pystan:Rhat for parameter mu[5] is 1.1329334583549957!
WARNING:pystan:Rhat for parameter mu[8] is 1.1012985603437124!
WARNING:pystan:Rhat for parameter mu[9] is 1.272914183469249!
WARNING:pystan:Rhat for parameter mu[10] is 1.113632904069064!
WARNING:pystan:Rhat for parameter mu[11] is 1.187569597575632!
WARNING:pystan:Rhat for parameter mu[12] is 1.1094679324389236!
WARNING:pystan:Rhat for parameter mu[13] is 1.4082115802696265!
WARNING:pystan:Rhat for parameter mu[14] is 1.276055315738049!
WARNING:pystan:Rhat for parameter mu[15] is 1.1087787800158826!
WARNING:pystan:Rhat for parameter mu[16] is 1.2963576993251207!
WARNING:pystan:Rhat for parameter mu[17] is 1.1164370441723843!
WARNING:pystan:Rhat for parameter mu[18] is 1.1314766207266838!
WARNING:pystan:Rhat for parameter mu[19] is 1.391710552807151!
WARNING:pystan:Rhat for parameter mu[20] is 1.2791575842645473!
WARNING:pystan:Rhat for parameter mu[21] is 1.3411289365980965!
WARNING:pystan:Rhat for parameter mu[22] is 1.1178469867881033!
WARNING:pystan:Rhat for parameter mu[25] is 1.415429818165475!
WARNING:pystan:Rhat for parameter mu[26] is 1.1080910996442717!
WARNING:pystan:Rhat for parameter mu[29] is 1.1754993455139824!
WARNING:pystan:Rhat for parameter mu[31] is 1.2929123072211073!
WARNING:pystan:Rhat for parameter mu[32] is 1.1178469867881033!
WARNING:pystan:Rhat for parameter mu[33] is 1.1115440695995307!
WARNING:pystan:Rhat for parameter mu[35] is 1.4101098585718697!
WARNING:pystan:Rhat for parameter mu[37] is 1.1807843590010574!
WARNING:pystan:Rhat for parameter mu[38] is 1.1469397405104818!
WARNING:pystan:Rhat for parameter mu[39] is 1.3656402421553926!
WARNING:pystan:Rhat for parameter mu[40] is 1.2033001436517838!
WARNING:pystan:Rhat for parameter mu[41] is 1.2317612848341253!
WARNING:pystan:Rhat for parameter mu[42] is 1.2238652725019374!
WARNING:pystan:Rhat for parameter mu[43] is 1.405887060970638!
WARNING:pystan:Rhat for parameter mu[44] is 1.268452519818485!
WARNING:pystan:Rhat for parameter mu[46] is 1.1845559142900657!
WARNING:pystan:Rhat for parameter mu[48] is 1.218781425377582!
WARNING:pystan:Rhat for parameter mu[49] is 1.1150323079747213!
WARNING:pystan:Rhat for parameter mu[50] is 1.1566569533578688!
WARNING:pystan:Rhat for parameter mu[51] is 1.414442026682911!
WARNING:pystan:Rhat for parameter mu[53] is 1.404715008587363!
WARNING:pystan:Rhat for parameter mu[54] is 1.4134917173056114!
WARNING:pystan:Rhat for parameter mu[55] is 1.248531745950521!
WARNING:pystan:Rhat for parameter mu[56] is 1.1967699213939647!
WARNING:pystan:Rhat for parameter mu[57] is 1.174879157718029!
WARNING:pystan:Rhat for parameter mu[58] is 1.203573734913662!
WARNING:pystan:Rhat for parameter mu[60] is 1.43051507380884!
WARNING:pystan:Rhat for parameter mu[62] is 1.208163963830866!
WARNING:pystan:Rhat for parameter mu[63] is 1.378089720872561!
WARNING:pystan:Rhat for parameter mu[64] is 1.2280987577661364!
WARNING:pystan:Rhat for parameter mu[65] is 1.3624647358564854!
WARNING:pystan:Rhat for parameter mu[66] is 1.2975351716107302!
WARNING:pystan:Rhat for parameter mu[67] is 1.3180397089494404!
WARNING:pystan:Rhat for parameter mu[69] is 1.3635360590019157!
WARNING:pystan:Rhat for parameter mu[70] is 1.3850330919377793!
WARNING:pystan:Rhat for parameter mu[71] is 1.310854771721129!
WARNING:pystan:Rhat for parameter mu[72] is 1.4415901772549387!
WARNING:pystan:Rhat for parameter mu[73] is 1.4704038776046267!
WARNING:pystan:Rhat for parameter mu[75] is 1.347066881596472!
WARNING:pystan:Rhat for parameter mu[76] is 1.4554099720540254!
WARNING:pystan:Rhat for parameter mu[77] is 1.4151632382158241!
WARNING:pystan:Rhat for parameter mu[78] is 1.3656665120674445!
```

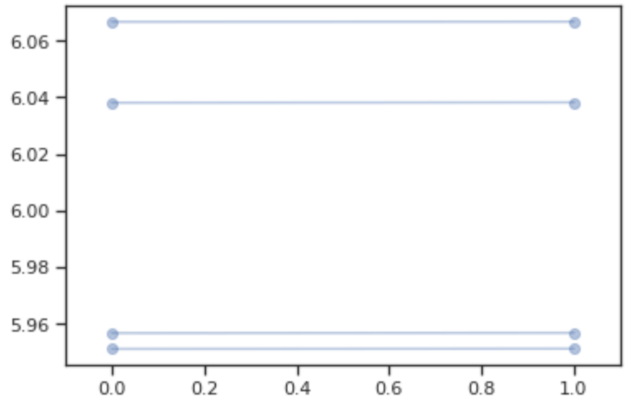
```
Rhat check :  False

WARNING:pystan:543 of 1000 iterations ended with a divergence (54.3%).
WARNING:pystan:Try running with adapt_delta larger than 0.8 to remove the dive
rgences.

N_eff check :  True
Divergence check:  False
```

In [92]:
```
sns.set(style="ticks")

a_sample = pd.DataFrame(varying_intercept_slope_sample['alpha'])
plt.figure(figsize=(14, 5))
sns.boxplot(data=a_sample, whis=np.inf, color="b")
plt.title("Samples of intercept per borough")
plt.show()

b_sample = pd.DataFrame(varying_intercept_slope_sample['beta'])
plt.figure(figsize=(14, 5))
sns.boxplot(data=b_sample, whis=np.inf, color = 'g')
plt.title("Samples of slope per borough")
plt.show()
```



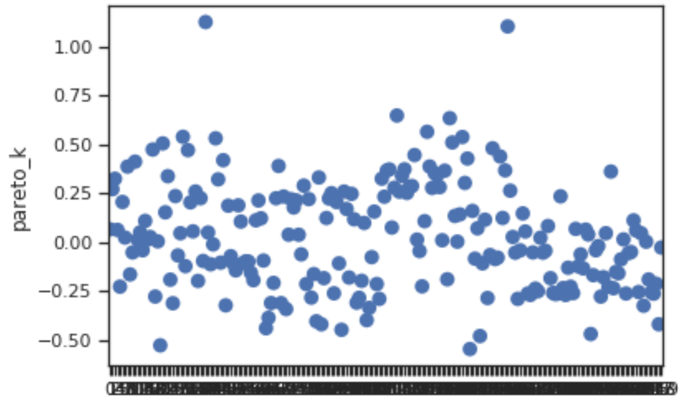


In [93]:
```
xvals = np.arange(2)
b = varying_intercept_slope_fit['alpha'].mean(axis=0)
m = varying_intercept_slope_fit['beta'].mean(axis=0)
for bi,mi in zip(b,m):
    plt.plot(xvals, mi*xvals + bi, 'bo-', alpha=0.4)
plt.xlim(-0.1, 1.1);
```



In [94]:
```
loglik = (varying_intercept_slope_sample["log_lik"])
psisloo = stanity.psisloo(loglik)
print("psisloo: ",psisloo.elpd)
psisloo.plot()
lppd_pooled = 0
for i in range (Y_train.shape[0]):
    lppd_pooled = lppd_pooled + np.log(np.mean(np.exp(loglik[:,i])))

peff_pooled = np.sum(lppd_pooled) - psisloo.elpd
print("p_eff: ", peff_pooled)
```

```
psisloo:  128.84553073286884
p_eff:  6.931903938751532
```



In [95]:
```
psisloo.print_summary()
```

Out[95]:
```
greater than 0.5     0.045455
greater than 1       0.009091
dtype: float64
```

In [ ]:
```
#Predict y_pred, not in use because model not diverge
"""y_pred = []
for i in range(len(Y_test)):
    boroughindx = X_test.Borough[i]
    y_pred.append(b[boroughindx] + m[boroughindx]*X_test.Enrollment[i])
print(round(MAE(Y_test, y_pred),3))
round(MSE(Y_test, y_pred),3)
plt.hist(y_pred, bins = 20)
"""
```

Hierarchical model

In [99]:
```
hierarchical_code = """
data {
    int<lower=1> N;
    int<lower=1> K;
    matrix[N, K] y;
}
parameters {
    real mu0;
    real<lower=0> sigma0;
    vector[K] mu;
    real<lower=0> sigma;
}
model {
    for (j in 1:K){
        mu[j] ~ normal(mu0, sigma0);
        y[:,j] ~ normal(mu[j], sigma);
     }
}
generated quantities {
    real mupred;
    mupred <- normal_rng(mu0, sigma0);
}
"""
```

In [100]:
```
data = []
for s in i_boroughs:
    np.random.shuffle(s)
    s = s[:min_len]
    data.append(Y.iloc[s])
```

In [101]:
```
data = np.array(data)
```

In [102]:
```
y_test = np.log(test_df.math)

data_hierarchical = {
    'N': data.shape[0],
    'K': data.shape[1],
    'y': data
}

fit_hierarchical = pystan.stan(model_code = hierarchical_code, data = data_hier
archical, iter=2000, chains=2)
mu_pred = np.mean(fit_hierarchical['mupred'])
sigma_pred = np.mean(fit_hierarchical['sigma'])
mu_ytest = np.mean(y_test)
sigma_ytest = np.std(y_test)
y_norm = (y_test - mu_ytest)/sigma_ytest
y_pred = (y_norm * sigma_pred) + mu_pred
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_bb063ad4e6e043a948b654
1ceedcd53e NOW.
/opt/conda/lib/python3.6/site-packages/Cython/Compiler/Main.py:367: FutureWarn
ing: Cython directive 'language_level' not set, using 2 for now (Py2). This wi
ll change in a later release! File: /tmp/tmpkp1cqxom/stanfit4anon_model_bb063a
d4e6e043a948b6541ceedcd53e_4686075826154491231.pyx
  tree = Parsing.p_module(s, pxd, full_module_name)
WARNING:pystan:4 of 2000 iterations ended with a divergence (0.2%).
WARNING:pystan:Try running with adapt_delta larger than 0.8 to remove the dive
rgences.
WARNING:pystan:Chain 1: E-BFMI = 0.05462743571074349
WARNING:pystan:Chain 2: E-BFMI = 0.061825775064211684
WARNING:pystan:E-BFMI below 0.2 indicates you may need to reparameterize your
model
```

In [103]:
```
print("Rhat check : ",pystan.diagnostics.check_rhat(fit_hierarchical))
print("N_eff check : ",pystan.diagnostics.check_n_eff(fit_hierarchical))
print("Divergence check: ", pystan.diagnostics.check_div(fit_hierarchical))
```

```
Rhat check :  True

WARNING:pystan:4 of 2000 iterations ended with a divergence (0.2%).
WARNING:pystan:Try running with adapt_delta larger than 0.8 to remove the dive
rgences.

N_eff check :  True
Divergence check:  False
```