Bayesian Models for Demographic Predictions

Abhilash Jain
Alexandru Dumitrescu

December 9, 2018

Contents

1 Introduction

1.1 The Problem:

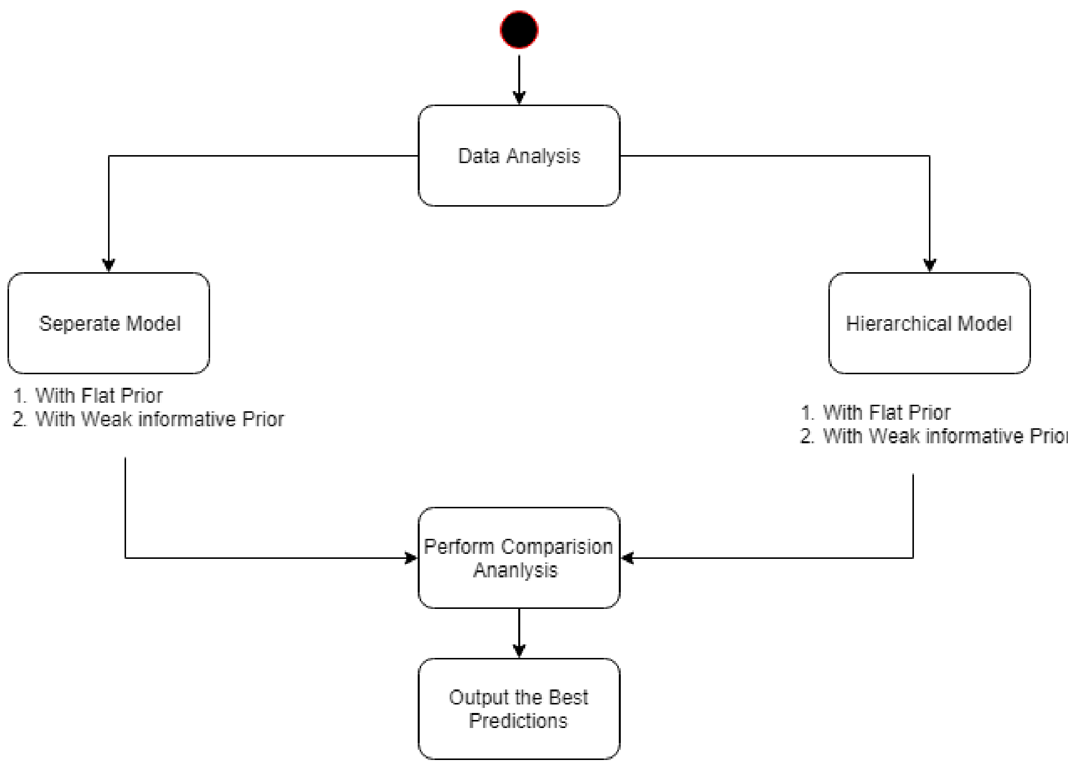
The Finnish Immigration Services are responsible for accessing citizenship applications and granting citizenships. As this is laborious and a time-consuming task, each application takes close to 6-8 months(sometimes more) to process and provide a decision. Future planning and prediction on this matter can help the Agency to better prepare and speed-up this process.

1.2 The Solution:

Currently we can access data from Migri regarding the demographical aspects of the people who are granted citizenships and our goal is to compare and assess the best model which can predict the number of granted citizenships for the upcoming year.

1.3 The Approach:

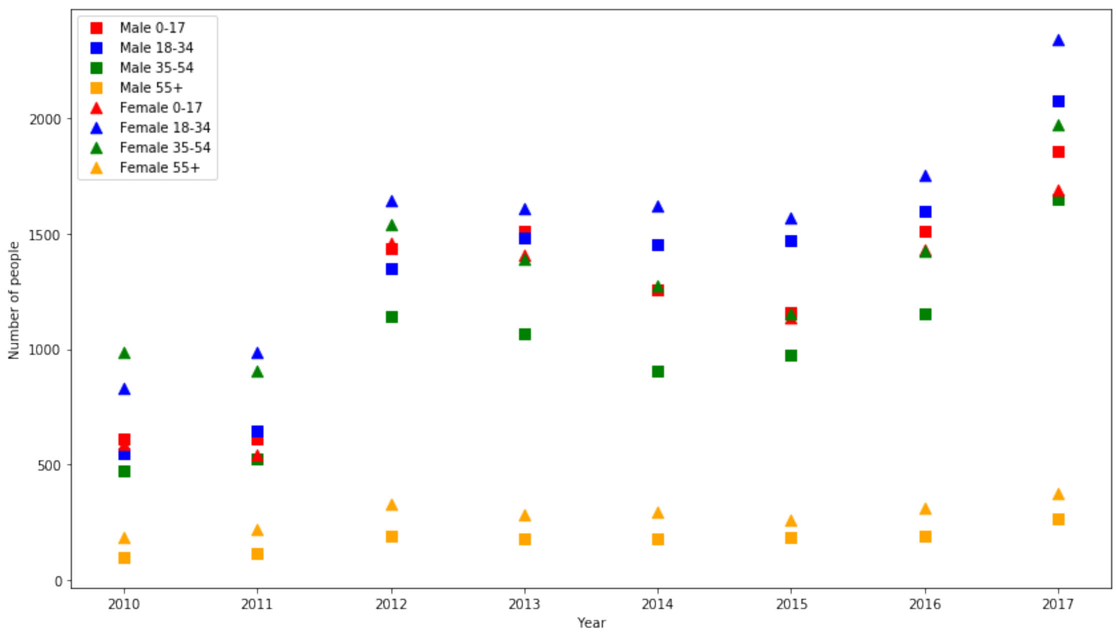
Below is a a flow-chart of our Approach for this Project:



2 Data Analysis

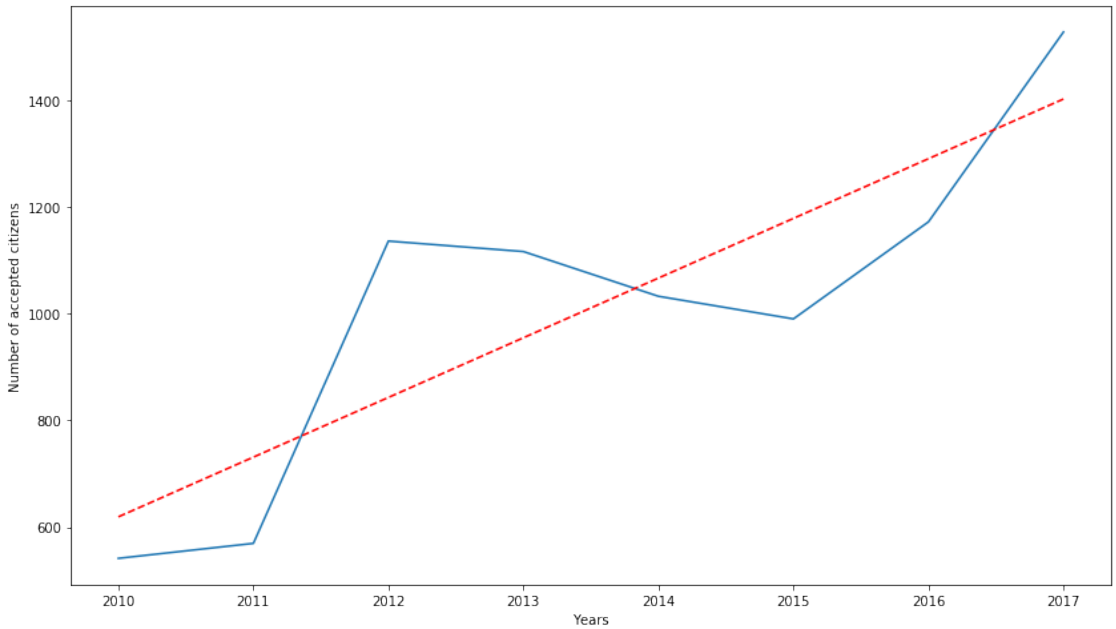
The dataset contains the number of granted Finnish citizenships over the years, based on age categories and gender. It can also be noticed that the acceptance number has an approximately increasing trend over the years, on each category, so our focus will be on linear regression models.

	Male				Female			
	0-17	18-34	35-54	55+	0-17	18-34	35-54	55+
2010	614	551	475	99	590	833	985	187
2011	613	645	527	118	544	985	904	222
2012	1438	1350	1142	190	1459	1642	1538	328
2013	1509	1483	1070	180	1409	1608	1388	283
2014	1260	1451	906	181	1266	1622	1277	297
2015	1162	1473	977	187	1139	1570	1154	259
2016	1513	1598	1156	194	1431	1751	1422	310
2017	1855	2074	1649	266	1689	2339	1971	376



The above scatter plot distinguishes between each age and gender category and we came up with the following comparisons between the Categories themselves:

- The Number of Granted citizenships is the highest for 'Young Adults' (between the ages of 18-34), which corresponds to the the need of workforce in the Finnish Market.
- The lowest Granted citizenships is the oldest Age category, signifying that the government doesn't want to provide and take a burden of an already aging population.



The above graph is the summarization of all the Accepted Number of people combined as one over the years 2010-2017. Two things clearly pop-out:

- We do see a spike in increase of number of granted citizenships from the year 2010 to 2017
- As this increase is quite satisfactory for a linear increasing trend, we will model the data on a linear regression for all the age categories individually. Concretely, the values for each year will be sampled as a normal distribution, around the mean point given by a regression line that is similar to the one that can be seen on the above graph. Moreover, this will be done for each age category for males and females, considering a regression line along the years for each one of the columns that appear in the dataframe printed above.

3 Models : Overview

From the conclusion of our Data Analysis, we will have models based on liner regression for each age category, separately. We will be using PyStan for all our models, and use Python to perform all operations on these models
More specifically, the models will be of the form:

- Samples for all our models will be drawn from a normal distribution:

$$y_i \sim N(\mu_i, \sigma_{G_i})$$

- Each μ_i will be a transformed parameter of some linear dependency between it and the years - again with the intuition of the approximately linearly increasing trend over the years:

$$\mu_i = \alpha_{G_i} \cdot years + \beta$$

Here, α_{G_i} represents the a specific slope of one of the 8 categories printed in the dataframe on data analysis. They will be taken either separately or from a hierarchy. To be more exact, the models we tried in this project were:

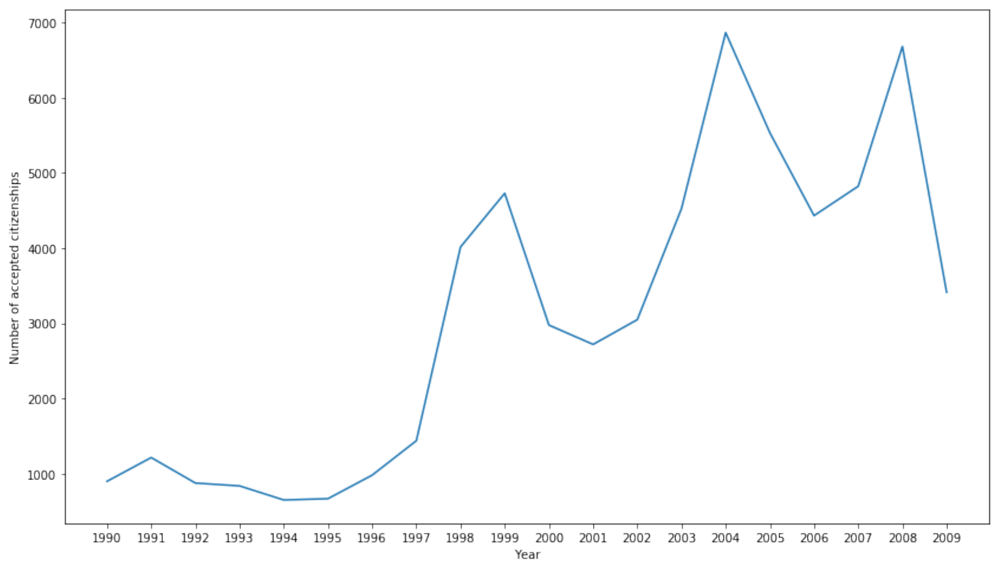
- Hierarchical models
  - Hierarchy on the slope α
  - Hierarchy on the constant β
  - Hierarchy on the variance parameter σ and α
- Separate model

4 Prior Analysis

For our Analysis we will be using the following priors for our models:

- Flat Priors
- Weakly Informative Priors on different hyper-parameters of the hierarchical model, or directly on the slope parameter for the regressions in the separate model

Flat priors are taken by default in the Stan Model, if we do not define any prior at all.
For the Weakly Informative we will analyze a Secondary dataset of the number of granted citizenships from the years 1990-2009. The difference between these two datasets is that, the second one does not have the numbers divided into age categories. The reason for taking such a prior is to "help" the model infer better, using what seems to be reasonable limits for the increase and decrease of the number of accepted Finnish citizenship from one year to another.



Above is the plot of the data points before 2010 and from this data we infer the following:

- **Mean increase over one year(μ_{prev}): 21.46857**
- **Mean standard deviation of these slopes(σ_{prev}): 168.0712268**

These values represent an average slope and standard deviation over each two consecutive years. To be more exact, the values from which we can see this mean and std are computed as follows:

$$diff[i] = \frac{No_{citizens}[year_i] - No_{citizens}[year_{i-1}]}{year_i - year_{i-1}}$$

with the denominator being obviously 1 at each point.

In the case of the prior for α parameter (applied on the separate model):

$$\alpha \sim N(\mu_{prev}, \sigma_{prev})$$

or, for the α-σ hierarchy model (hierarchy on both parameters):

$$\mu_{0-alpha} \sim N(\mu_{prev}, \sigma_{prev})$$

We will come to this later, but for the α-σ hierarchy model, $\mu_{0-alpha}$ represents the mean normal parameter for the hierarchy on the slope α, so the priors will be applied on the hierarchy's hyperparamter.

5 Models: An In-Depth Analysis

5.1 Separate model with flat prior

The first model we tried out is a separate model with linear regressions for each category (from now on, consider a category as being one of the 8 total categories, coming from both male and female, as this is how the models were made). First, no prior will be taken into account (flat prior).

The model is:

$$y_i \sim N(\mu_i, \sigma_{G_i})$$

With σ_{G_i} being specific for each category and a μ_i taken at each data point from it's respective group's regression line:

$$\mu_i = \alpha_{G_i} \cdot years + \beta_{G_i}$$

The regression lines are inferred by taking the number of accepted citizens as a function of years.

Please find the Stan Code for the model: Link to model

5.1.1 Convergence diagnostics

All the parameters for the model seem to have converged, since all the $\hat{\mathbf{R}}$ values are very close to 1 and the effective sampling size $\mathbf{n_{eff}}$ seems to be reasonable. There are no parameters that diverged (for this, or any other model).

Convergence for the α parameter

	alpha[0]	alpha[1]	alpha[2]	alpha[3]	alpha[4]	alpha[5]	alpha[6]	alpha[7]
n_eff	692.0	510.0	657.0	552.00	1098.0	849.00	854.0	682.0
Rhat	1.0	1.0	1.0	1.0	1.0	1.01	1.0	1.0

Convergence for the β parameter

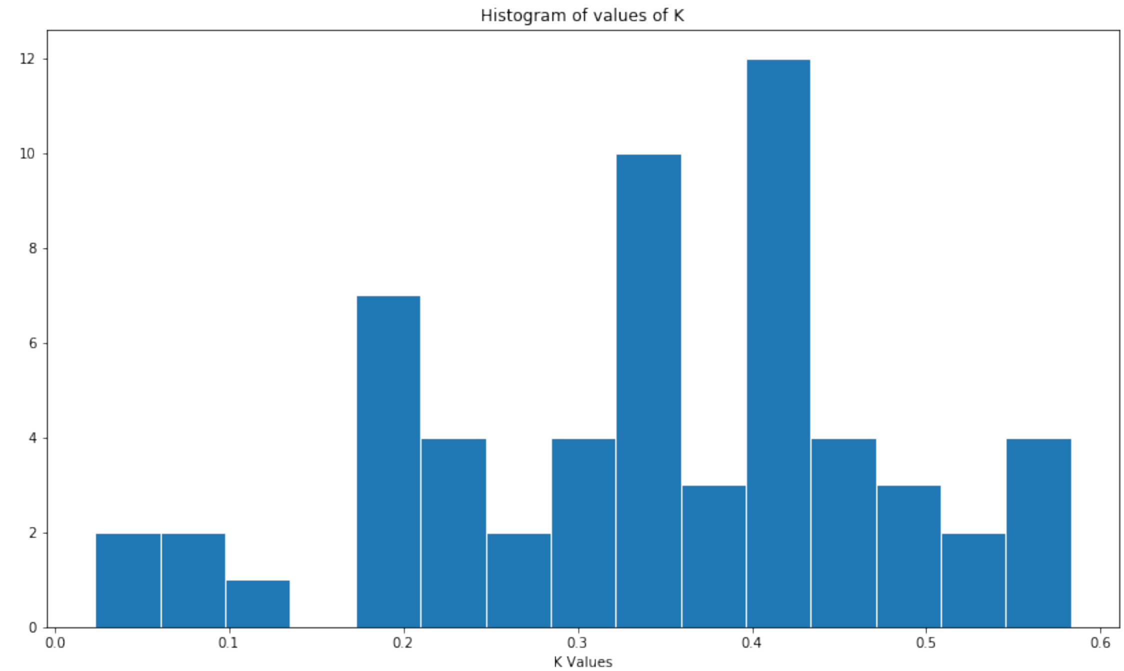
	beta[0]	beta[1]	beta[2]	beta[3]	beta[4]	beta[5]	beta[6]	beta[7]
n_eff	692.0	510.0	657.0	552.00	1098.0	849.00	854.0	682.0
Rhat	1.0	1.0	1.0	1.01	1.0	1.01	1.0	1.0

Convergence for the σ parameter

	sigma[0]	sigma[1]	sigma[2]	sigma[3]	sigma[4]	sigma[5]	sigma[6]	sigma[7]
n_eff	760.0	677.0	710.0	862.00	985.0	775.0	797.0	764
Rhat	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

K values and log psis measurements

Our k values turn out to be below 0.7, so our psis value will be a reliable assessment for the model.



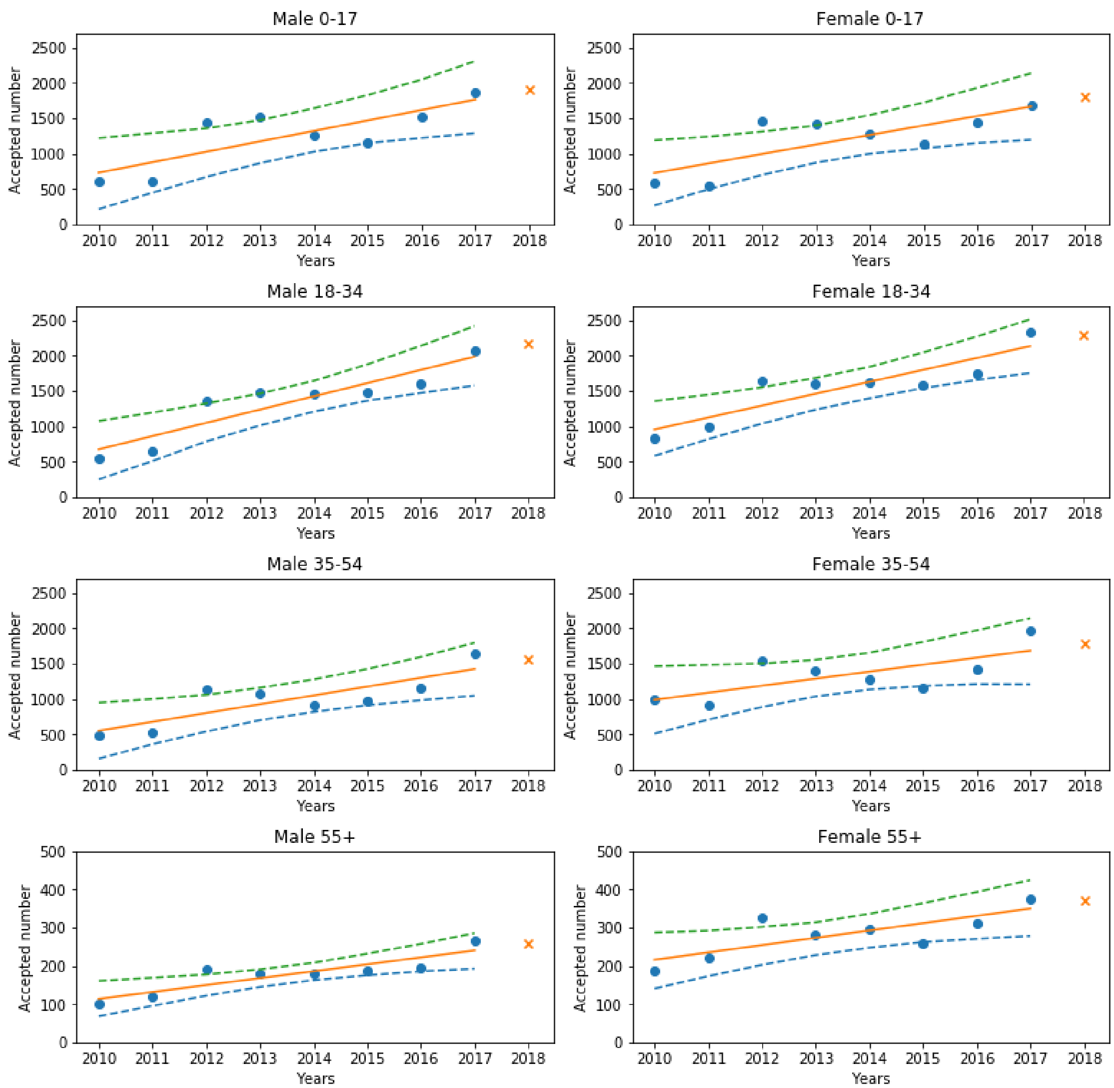
Psis value is -429.85609031128047

Peff value is 16.994522836727015

Not much to say about the Psis and Peff values now, but they will be taken later into consideration for model comparison, since the k values show that our measurement is reliable.

5.1.2 Predicting for the year 2018

Below is the plot containing the regression lines for each category (each α and β). The green and blue dotted lines represent the95% central interval for our μ values, sampled from a normal distribution around that line. The x mark at the end represents the mean predicted value for 2018.

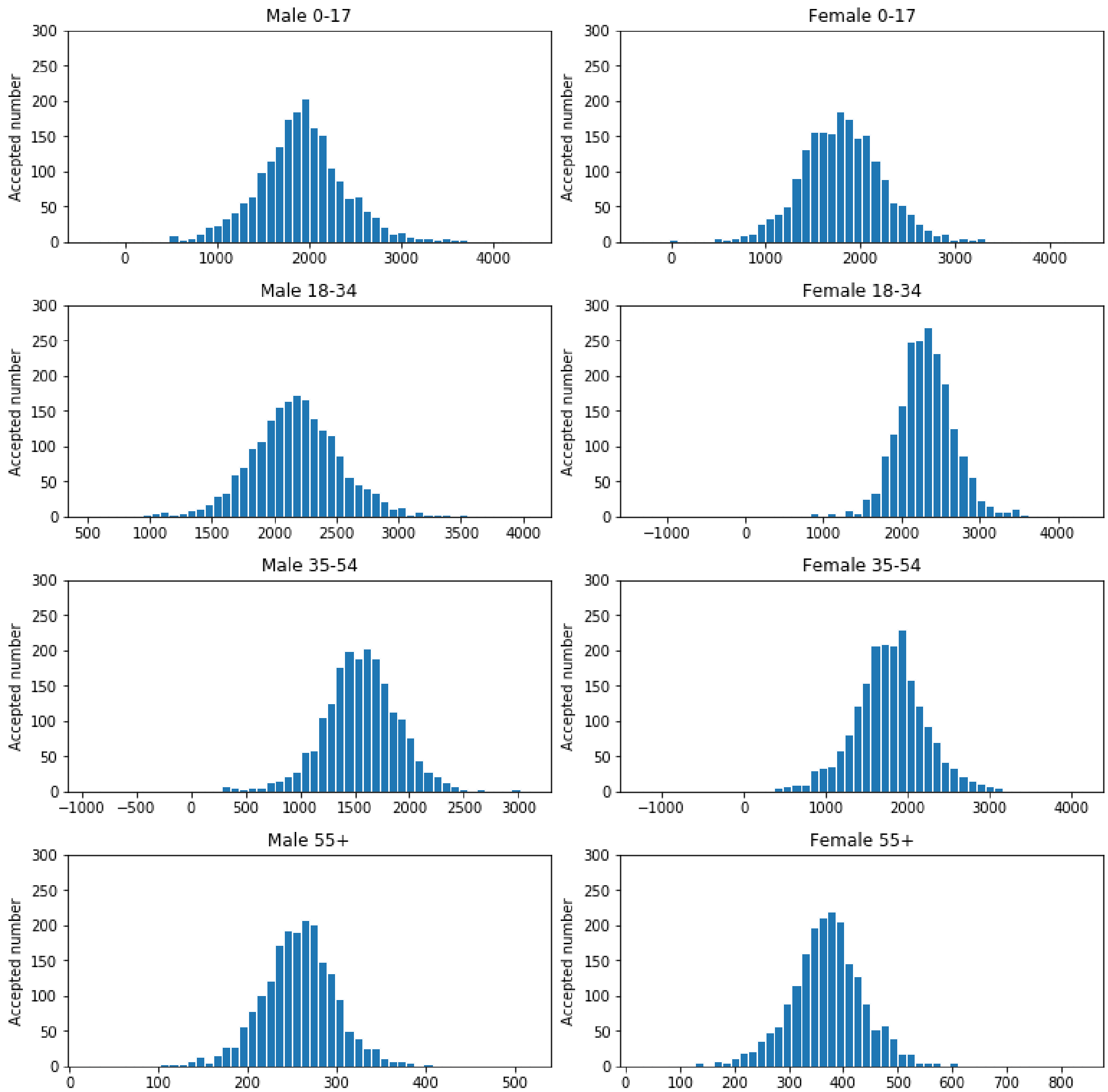


5.1.3 Analysis of the Results

The plots look reasonable. The regression lines seem to fit the data as best as a first order polynomial line would be able to. The 95% interval is fitting the line somewhat tightly to that line, but more importantly, since we have different values for each group, the data points seem to be nicely contained in these 95% intervals. For instance, the variance for the elderly is much lower (in the data) compared to the other younger groups, and that is reflected as well by the model. The mean prediction for 2018 is obviously, as expected, to be approximately on the regression line.

Also, note our previous definition of the model, in which we wrote $y_i \sim N(\mu_i, \sigma_{G_i}$, with $\mu_i = \alpha_{G_i} \cdot years + \beta_{G_i}$. What all the models will do in this problem, is find a regression line from which μ_i will be taken, dependent of years, and the actual sampling will be done together with an individual std σ, for each group.

5.1.4 Post Predictive Analysis



The predictions for 2018 are, as the model described, normally sampled from our model with a normal distribution, $y_i \sim N(\mu_i, \sigma_{Gi})$, with a mean around the x point plotted above. Visualizing these plots is similar to drawing the dotted points from above, in the direction they were heading, and imagining the points being around that x point, with 95% of the samples contained between the dotted lines.

5.2 Separate Model with normal prior on the α parameter

In this section we will be defining our first model with a prior, which is defined from the section: **Prior Analysis**.The prior will be on the α - Parameter of the Liner Regression Model (The slope). So the model will look exactly the same, but with a prior (based on the calculations on the prior choosing section) on α:

$$\alpha \sim N(21.46, 168)$$

Please find the Stand Code for the model: Link to model

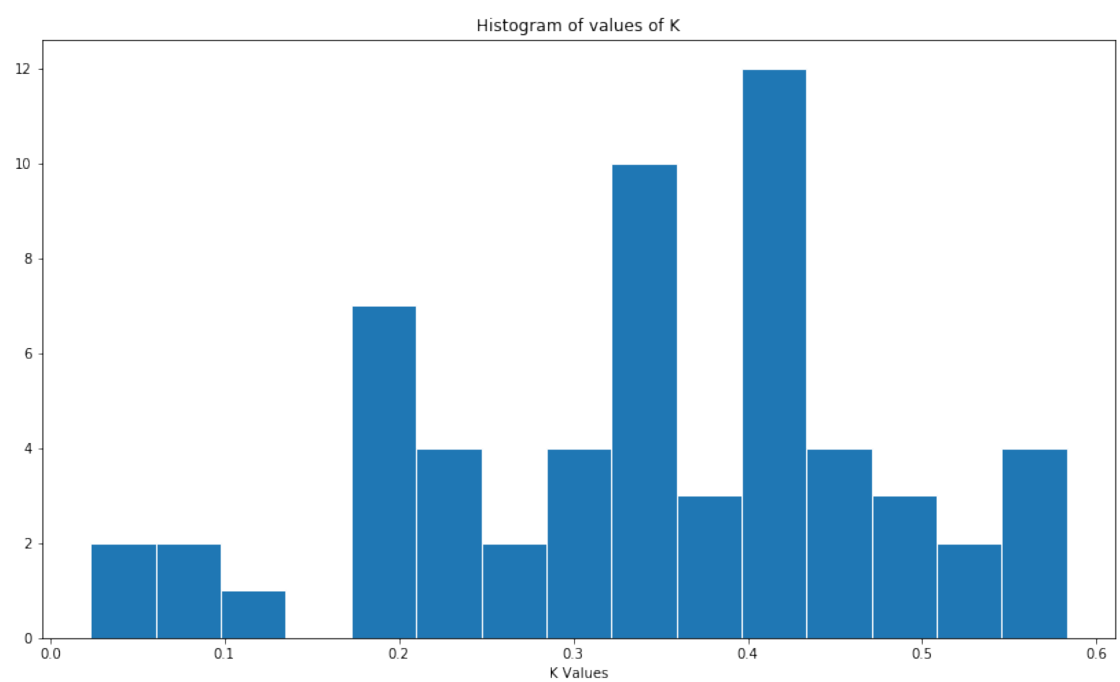
5.2.1 Convergence analysis

	alpha[0]	alpha[1]	alpha[2]	alpha[3]	alpha[4]	alpha[5]	alpha[6]	alpha[7]
n_eff	744.0	613.0	1090.0	600	807.0	961.0	892.0	871.0
Rhat	1.0	1.0	1.0	1.01	1.01	1.0	1.0	1.0

	beta[0]	beta[1]	beta[2]	beta[3]	beta[4]	beta[5]	beta[6]	beta[7]
n_eff	743.0	613.0	1090.0	600	807.0	961.0	892.0	871.0
Rhat	1.0	1.0	1.0	1.01	1.01	1.0	1.0	1.0

	sigma[0]	sigma[1]	sigma[2]	sigma[3]	sigma[4]	sigma[5]	sigma[6]	sigma[7]
n_eff	428.0	729.0	1003.0	699.00	708.0	1206.0	847.0	947.0
Rhat	1.0	1.0	1.0	1.0	1.0	1.0	1.01	1.0

K values and log psis measurements



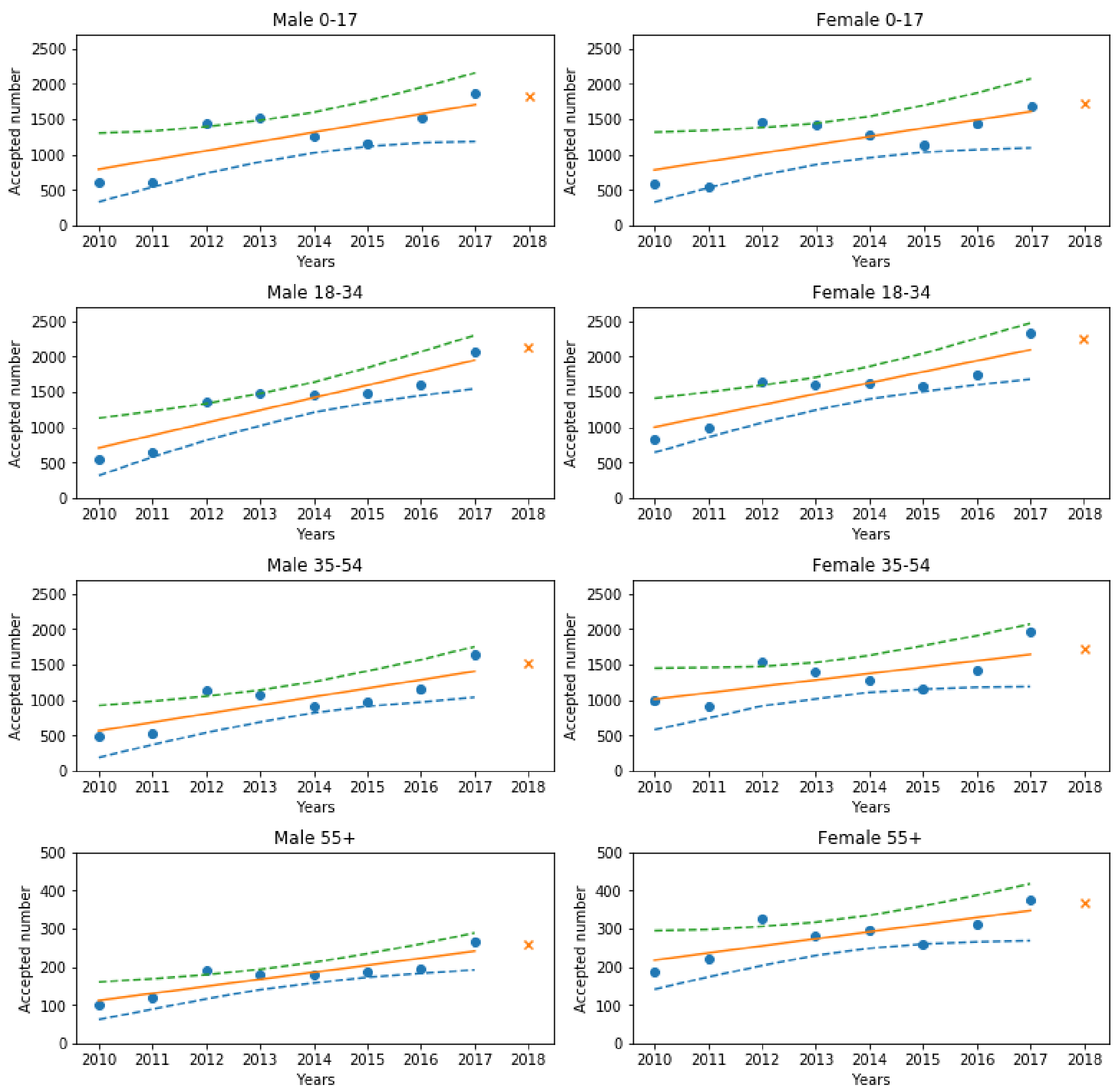
The Psis value is -430.30148

Peff values is 17.4035

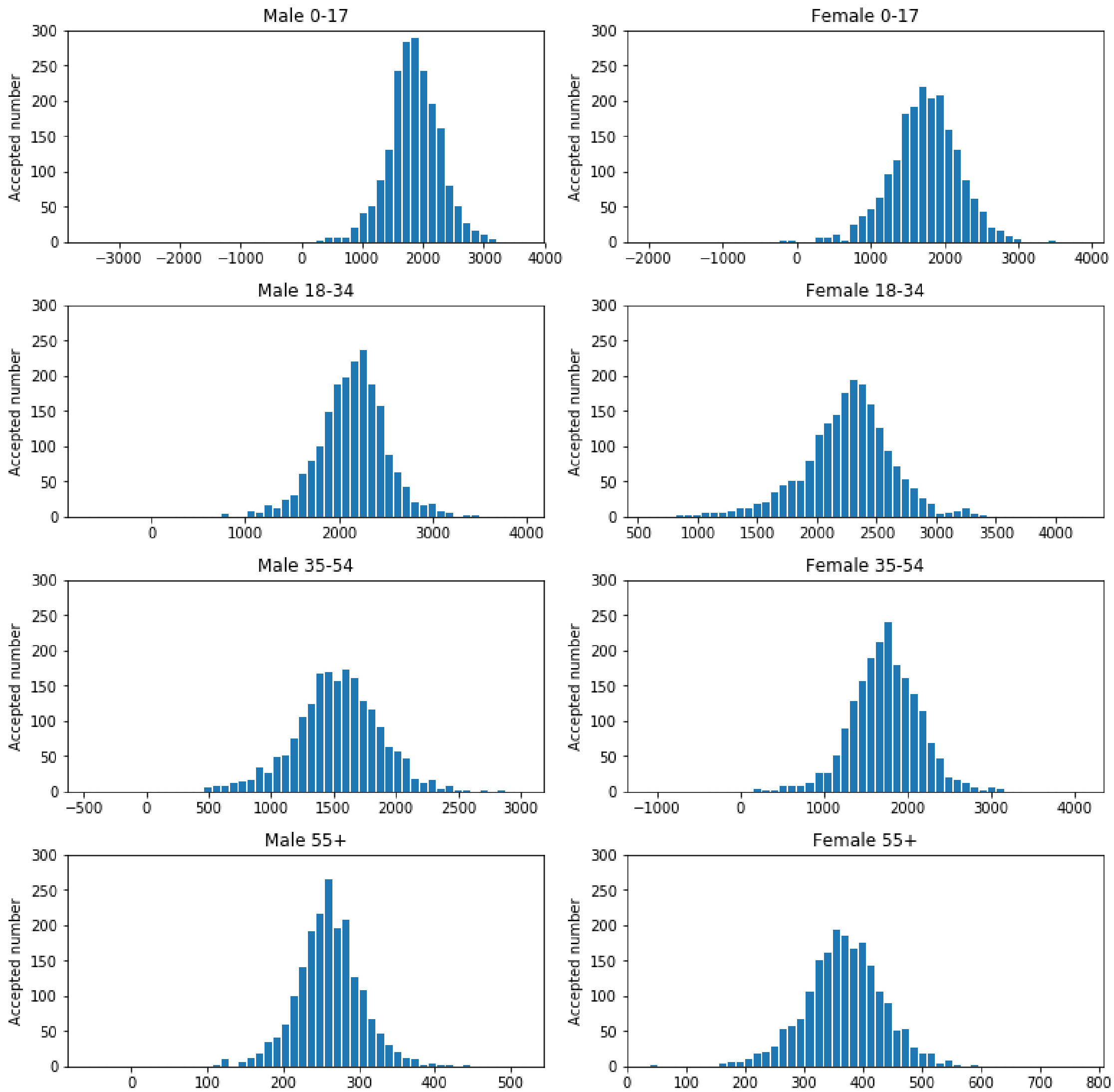
Since the k values are below 0.7, we can use the Psis values for comparison. Only based on these, the model seems to actually perform a bit worse than before.
A quick analysis on these: with respect to the model having a flat prior (the same model described in the previous section), which had -429.856 and 16.994 for Psis and Peff respectively, it seems that the model performed just a tiny bit worse, which would suggest a pretty low prior sensitivity.

5.2.2 Predicting for the year 2018

Next, the same kind of plots for the regression lines among with the dotted 95% around them and prediction samples for 2018 will be plotted.



5.2.3 Post Predictive Analysis



Analysis: One key insight from the above plots is that they are pretty similar to the ones for model with the flat prior, suggesting it's low prior sensitivity,

5.3 Hierarchical model for the slope α

The first hierarchical model we tried out is one with a hierarchy on α parameter. Which is defined as:

$$\alpha_{Gi}|\mu_0, \sigma_0 \sim N(\mu_0, \sigma_0)$$

α_{Gi} denotes the slope parameter corresponding to the specific group that data point i belongs to, just like in the separate model. The μ_i parameter is taken the same as before, from 8 α_{Gi} parameters (that are now grouped in a hierarchy), and then, finally, a normal sample with mean around our regression line is taken, i.e. (note that the model has the same form, with just an added slope hierarchy):

$$y_i \sim N(\mu_i, \sigma_i)$$

$$\mu_i = \alpha * years + \beta$$

Please find the Stand Code for the model: Link to model

5.3.1 Convergence analysis

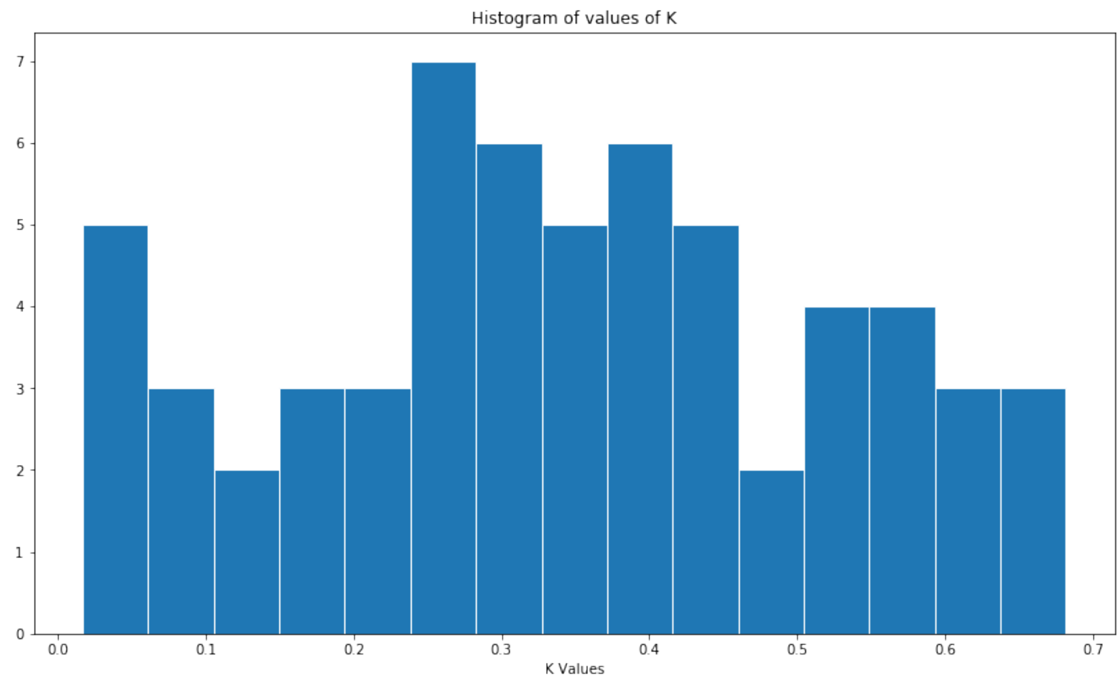
	alpha[0]	alpha[1]	alpha[2]	alpha[3]	alpha[4]	alpha[5]	alpha[6]	alpha[7]
n_eff	965.0	820.0	631.0	809.00	904.0	617.0	861.0	912.0
Rhat	1.01	1.01	1.0	1.0	1.0	1.0	1.0	1.0

	beta[0]	beta[1]	beta[2]	beta[3]	beta[4]	beta[5]	beta[6]	beta[7]
n_eff	965.0	820.0	631.0	809.00	904.0	617.0	861.0	912.0
Rhat	1.01	1.01	1.0	1.0	1.0	1.0	1.0	1.0

	sigma[0]	sigma[1]	sigma[2]	sigma[3]	sigma[4]	sigma[5]	sigma[6]	sigma[7]
n_eff	1012.0	823.0	953.0	745.00	861.0	406.0	990.0	1099.0
Rhat	1.0	1.0	1.0	1.0	1.0	1.01	1.0	1.0

	mu0	sigma0
n_eff	932	942
Rhat	1.0	1.0

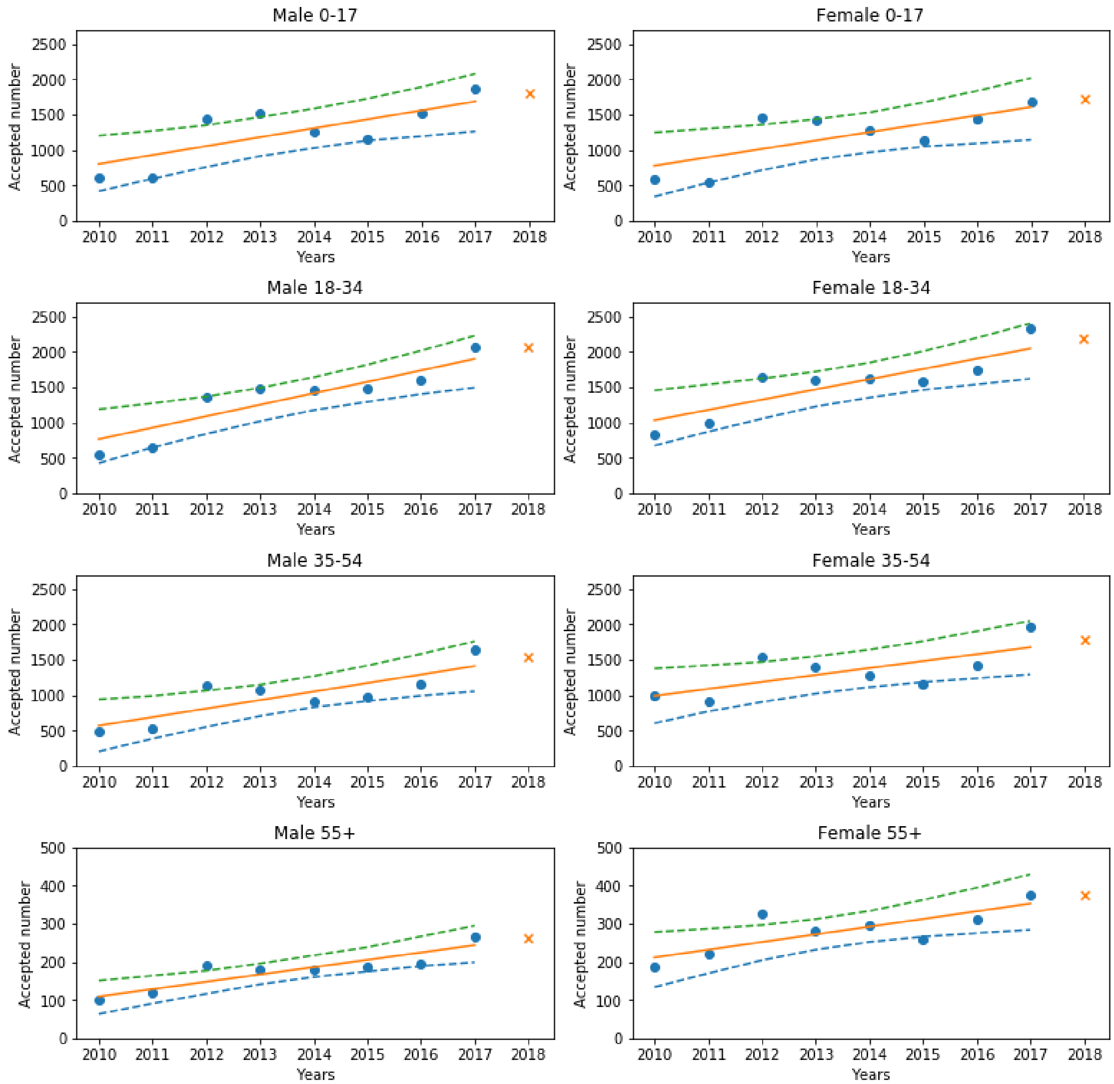
A quick analysis shows that the model converged, with $\hat{R}$ values close to 1 and reasonable n_{eff} values, also for the hyperparamters on the slope α **K values and log psis measurements**



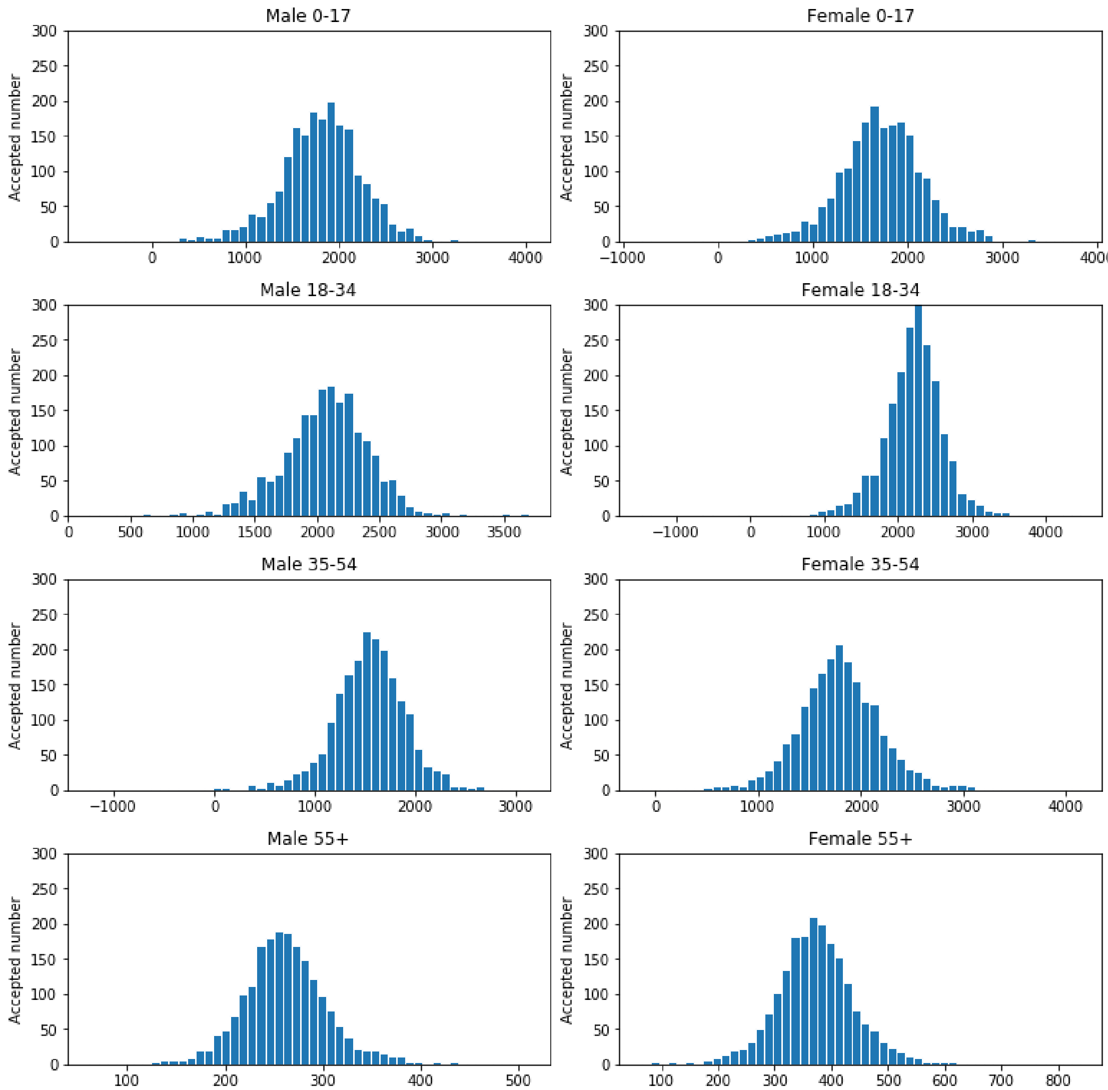
Again, our k values are below 0.7, which means that our model can be compared to the previous, separate one. Psis and Peff values are **-429.3181** and 16.5892, respectively. Not really an improvement compared to -429.85 and 16.99 values for the separate model with no priors, (which is best one until now.)

5.3.2 Predicting for the year 2018

Next, the same plots will be made, indicating the regression lines with their 95% intervals and histograms of the 2018 predicted values.



5.3.3 Post Predictive Analysis



Analysis of the Results: Again, doing a quick analysis, the model doesn't seem to do much better than the separate one. No additional priors will be carried out for this model (or the hierarchical β one, as we'll focus more on the hierarchy on two different parameters, $\alpha - \sigma$, which gave a significant improvement.

5.4 Hierarchical model for β

Next model is pretty similar to the hierarchical on α, but now the hierarchy will be on the constant value of the regressions, β:

$$\beta_{Gi} | \mu_{0beta}, sigma_{0beta} \sim N(\mu_{0beta}, sigma_{0beta})$$

Similar to the previous model, β_{Gi} represents the constant parameter for the regression belonging to the age group corresponding to that data point i.
Please find the Stand Code for the model: Link to model

5.4.1 Convergence analysis

	alpha[0]	alpha[1]	alpha[2]	alpha[3]	alpha[4]	alpha[5]	alpha[6]	alpha[7]
n_eff	1162.0	668.0	1067.0	789.00	1050.0	1275.0	1256.0	1058.0
Rhat	1.0	1.01	1.0	1.01	1.0	1.0	1.0	1.0

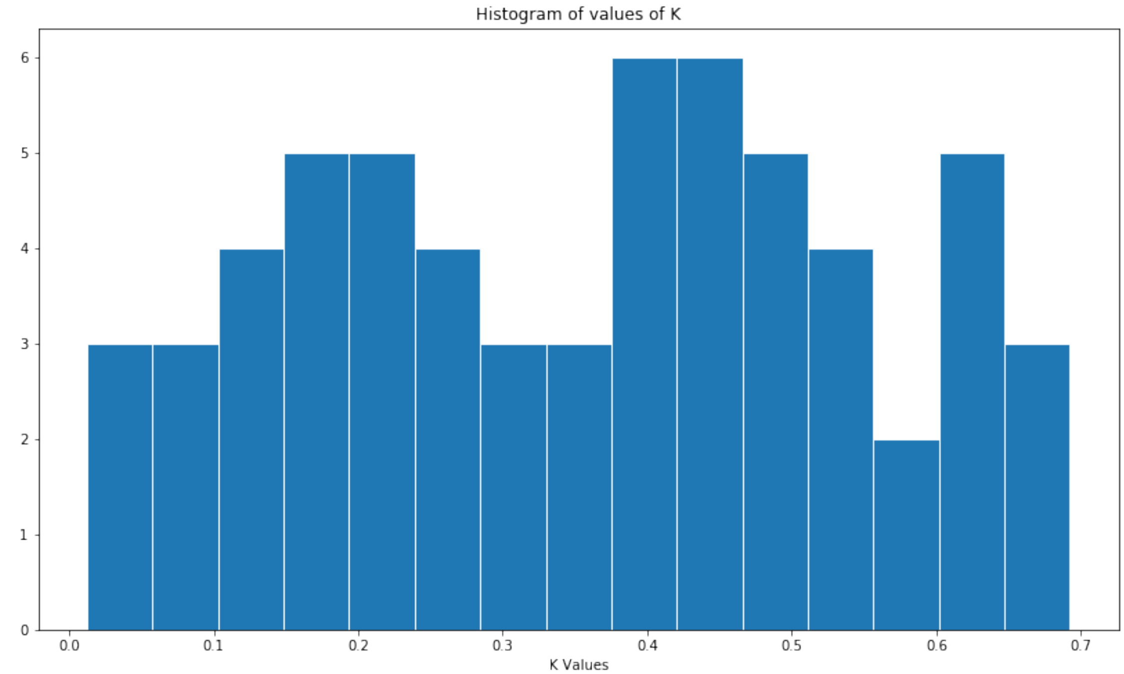
	beta[0]	beta[1]	beta[2]	beta[3]	beta[4]	beta[5]	beta[6]	beta[7]
n_eff	1163.0	668.0	1067.0	789.00	1050.0	1276.0	1256.0	1058.0
Rhat	1.0	1.0	1.0	1.01	1.0	1.0	1.0	1.0

	sigma[0]	sigma[1]	sigma[2]	sigma[3]	sigma[4]	sigma[5]	sigma[6]	sigma[7]
n_eff	1127.0	853.0	1163.0	894.00	977.0	1059.0	1319.0	1082.0
Rhat	1.0	1.0	1.0	1.0	1.0	1.01	1.0	1.0

	mu0_beta	sigma_beta
n_eff	1436	967.0
Rhat	1.0	1.0

This model also converged, as the values show.

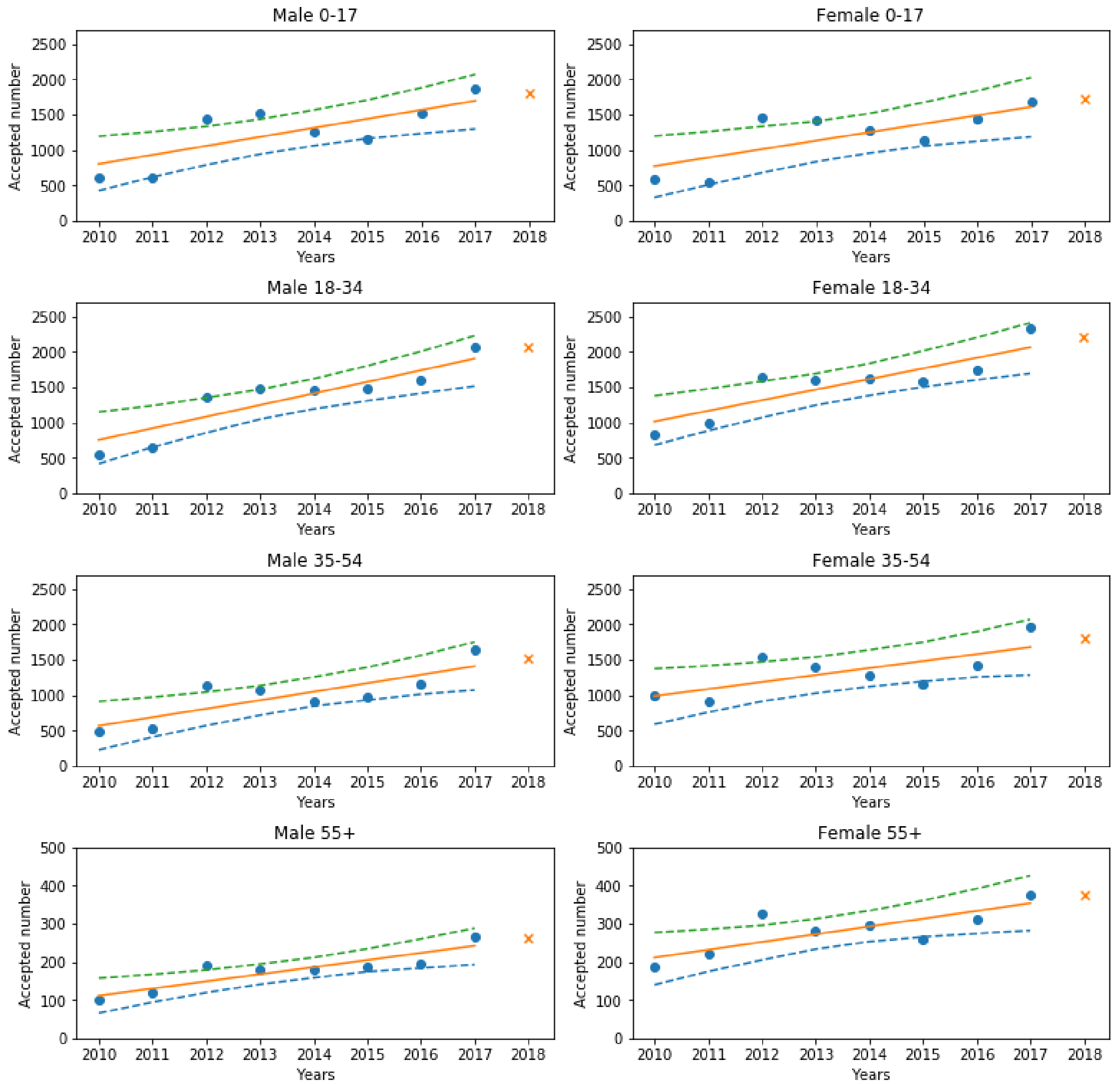
K values and PSIS measurements



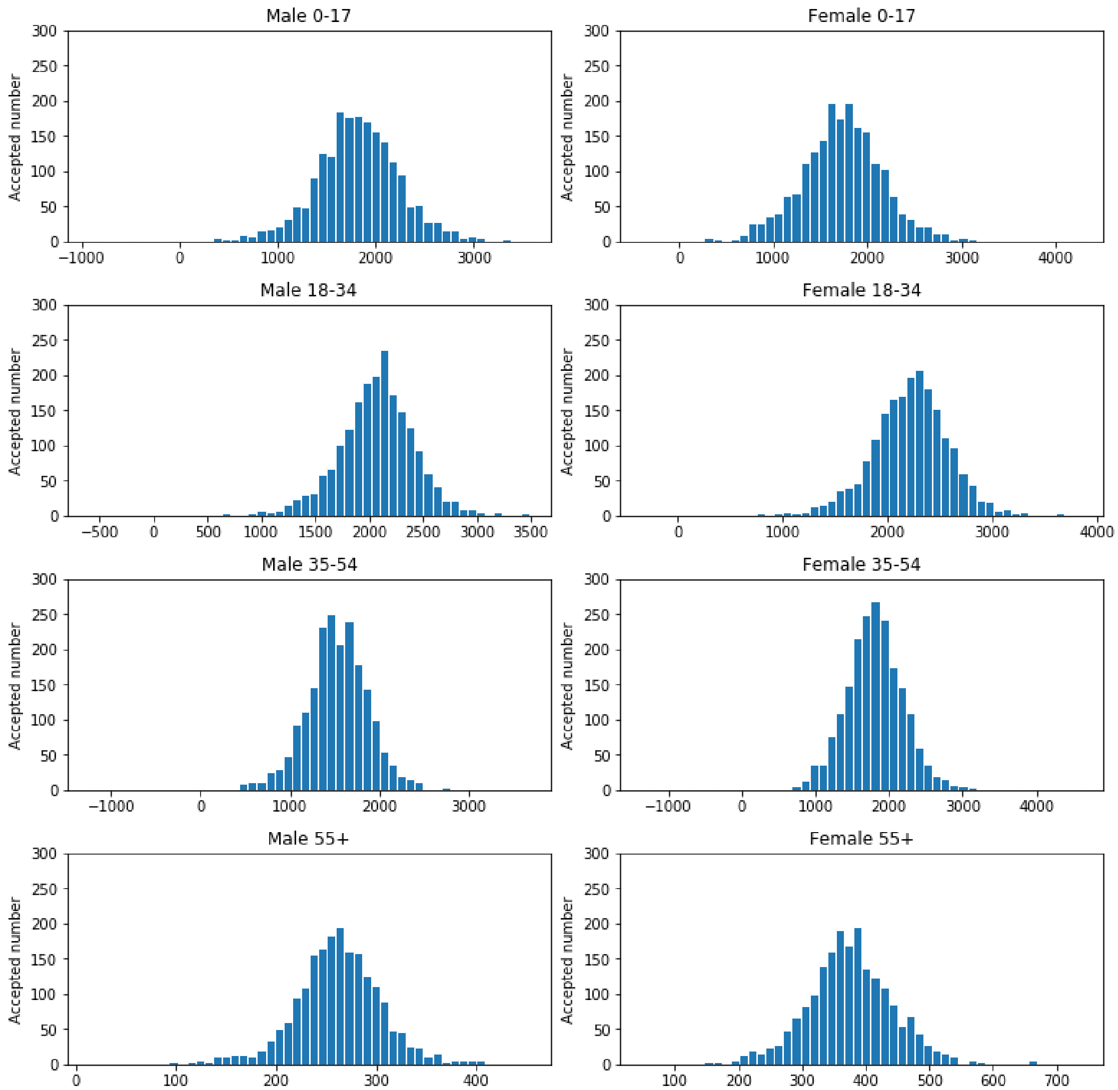
Again, this model also shows decent margin for k values, so it will also be taken into consideration for comparisons. The Psis and Peff -428.7834 and 16.3881. This is the first slight improvement compared to the separate model. Once again though, other priors won't be considered for this model.

5.4.2 Predictions for the year 2018

Next, plots for the regression lines and sampled predictions for 2018 are shown and are quite similar to the previous models



5.4.3 Post Predictive Analysis



Analysis: From the above histograms it is pretty clear that they are similar to the model before, but with better results on the PSIS and Peffective values

5.5 Hierarchical model for α and σ with a flatprior

This model is the one which gave the best results, and so, it will have the most rigorous prior analysis. Again, to define the model:

$$y_i \sim N(\mu_i, \sigma_{G_i})$$

The final target sample is the same as before. Now, compared to the other models, hierarchies on both the slopes α and the group variance σ will be applied.

$$\alpha_{G_i}|\mu_{0alpha}, \sigma_{0alpha} \sim N(\mu_{0alpha}, \sigma_{0alpha})$$

$$\sigma_{G_i}|p_1, p_2 \sim Cauchy(p_1, p_2)$$

Please find the Stand Code for the model: Link to model

5.5.1 Convergence analysis

	alpha[0]	alpha[1]	alpha[2]	alpha[3]	alpha[4]	alpha[5]	alpha[6]	alpha[7]
n_eff	1100.0	1466.0	1423.0	1138.00	1595.0	1429.0	1370.0	255.0
Rhat	1.0	1.01	1.0	1.0	1.0	1.0	1.0	1.01

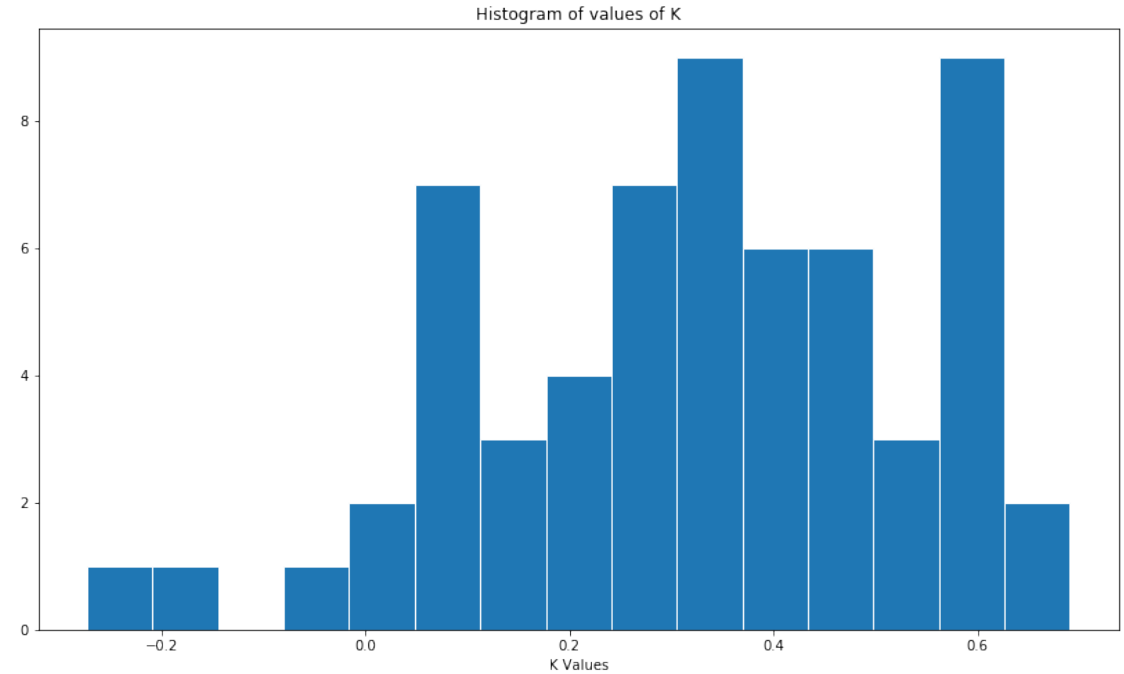
	beta[0]	beta[1]	beta[2]	beta[3]	beta[4]	beta[5]	beta[6]	beta[7]
n_eff	1100.0	1465.0	1423.0	1138.0	1595.00	1429.0	1371.0	255.0
Rhat	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.01

	sigma[0]	sigma[1]	sigma[2]	sigma[3]	sigma[4]	sigma[5]	sigma[6]	sigma[7]
n_eff	1193.0	1349.0	1157.0	740.00	1102.0	1296.0	1457.0	1082.0
Rhat	1.0	1.0	1.0	1.0	1.01	1.0	1.0	1.01

	mu0_alpha	sigma0_alpha	p1	p2
n_eff	1261.0	967.0	1317.0	368.0
Rhat	1.0	1.0	1.0	1.01

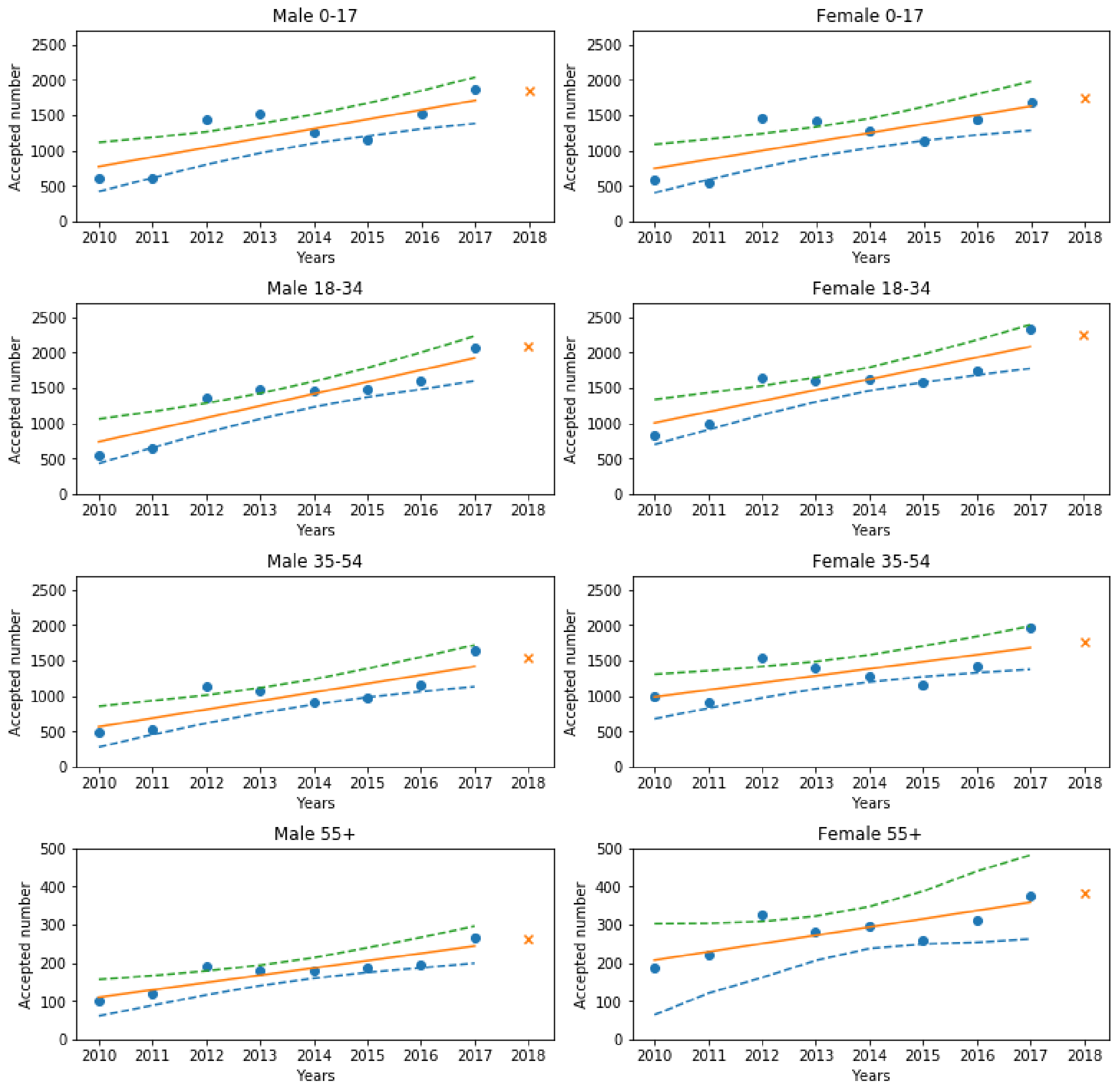
The model converged, and arguably more efficient, with higher number of effective samples taken, which already shows signs of improvement.

K values and log psis measurements



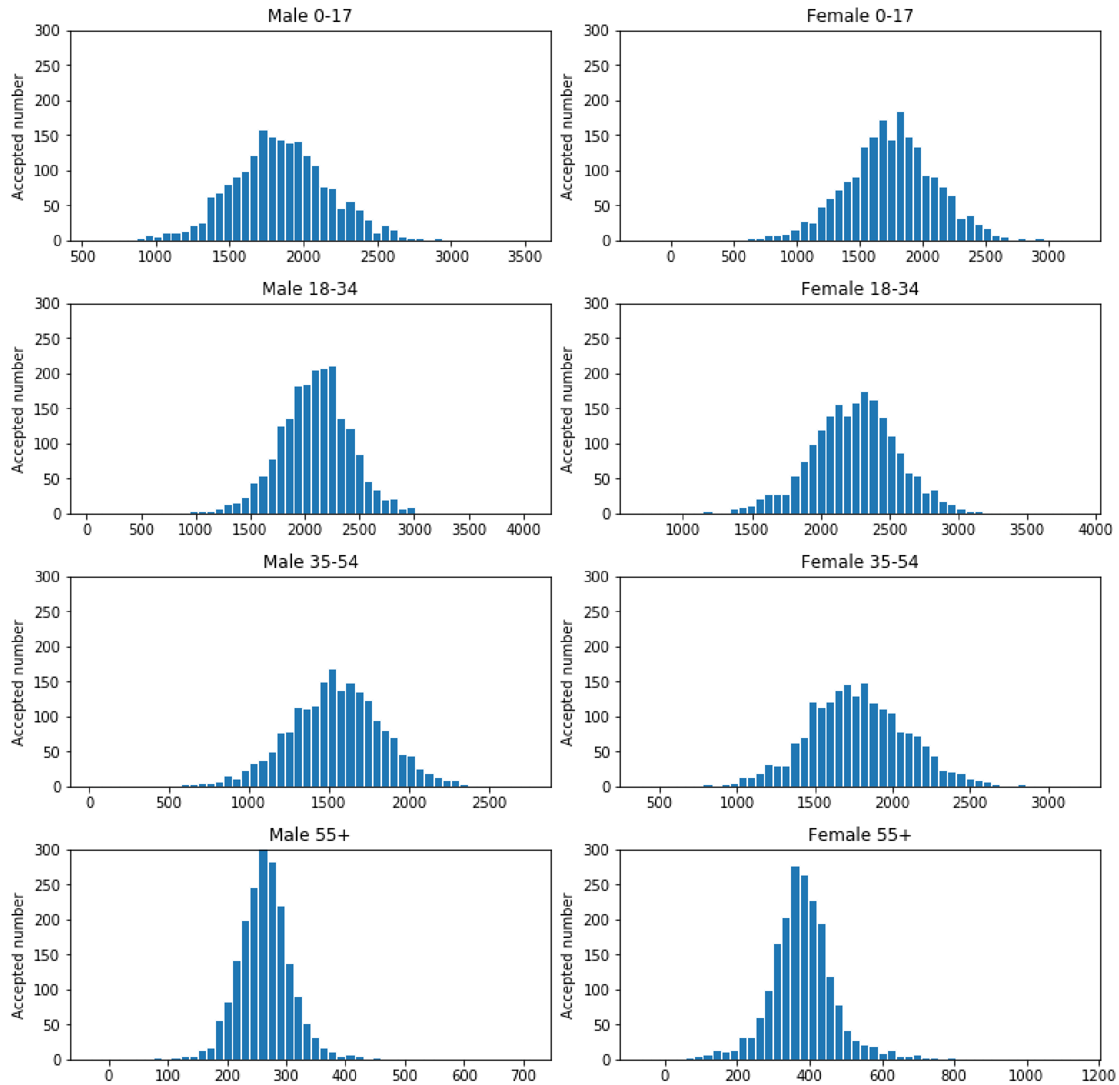
The K values are below 0.7, it allows us to confidently say that the model, at least on the error's perspective, it performs better than the separate and previous hierarchies tried for this data, having Psis and Peffective values of -426.9101 and 17.1181.

5.5.2 Predicting results for 2018



We finally have an actual visual improvement, particularly in "Female 55+" category. The variance taken for this category, seems to fit quite a bit better all the data points for this specific age group, whereas before, the model had a much more skewed variance, similar to what can be seen on the left, for the "Male 55+".

5.5.3 Post Predictive Analysis



Again, histograms of the samples taken from 2018. This time, the model seems to have performed more accurately on the elder group and has quite a bit better margins on the other categories as well. For this model, since we consider it to be the best one we got, will have a couple of different priors, to check prior sensitivity and try to get even better results.

5.6 Hierarchical model for α and σ and normal, more informative prior on $\mathbf{mu_{0alpha}}$

The model is exactly the same as before, with the only difference that the prior for the mean hyper-parameter for α has lower variance:

$$\mu_{0alpha} \sim N(21.46, 168)$$

These values represent the exact values that we got from the data containing the previous years.

Please find the Stand Code for the model: Link to model

5.6.1 Convergence analysis

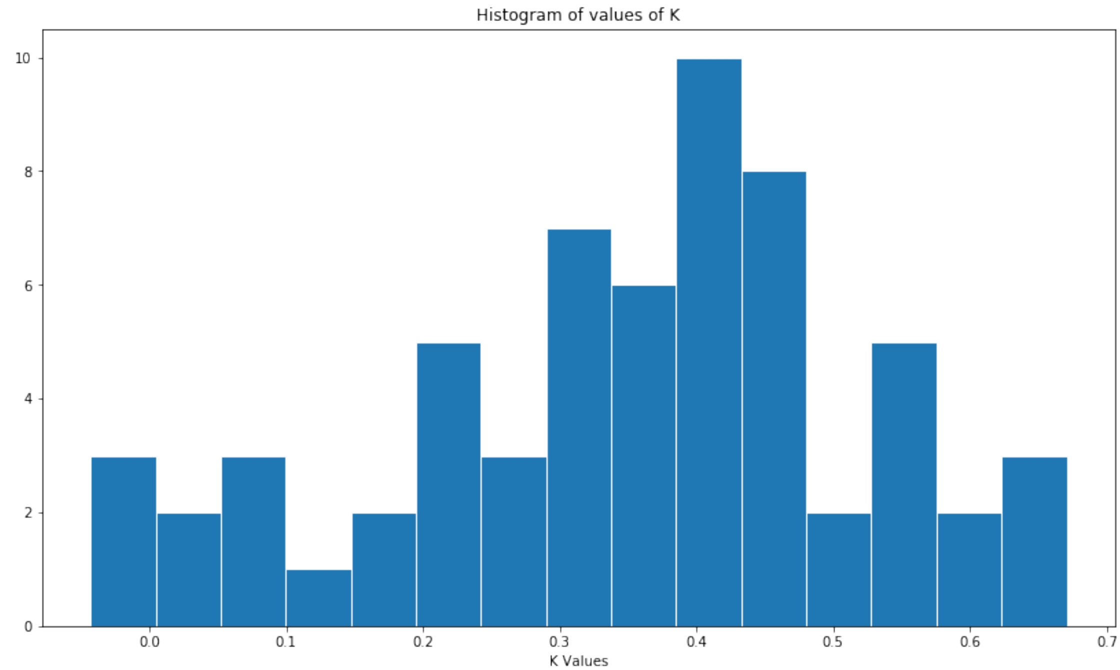
	alpha[0]	alpha[1]	alpha[2]	alpha[3]	alpha[4]	alpha[5]	alpha[6]	alpha[7]
n_eff	1180.0	1243.0	1512.0	1039.00	1429.0	1338.0	1177.0	320.0
Rhat	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.01

	beta[0]	beta[1]	beta[2]	beta[3]	beta[4]	beta[5]	beta[6]	beta[7]
n_eff	1180.0	1243.0	1512.0	1039.00	1429.0	1338.0	1177.0	320.0
Rhat	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.01

	sigma[0]	sigma[1]	sigma[2]	sigma[3]	sigma[4]	sigma[5]	sigma[6]	sigma[7]
n_eff	1310.0	1179.0	1139.0	727.00	1044.0	1532.0	1233.0	292.0
Rhat	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.02

	mu0_alpha	sigma0_alpha	p1	p2
n_eff	2000.0	1097.0	1386.0	629.0
Rhat	1.0	1.0	1.0	1.01

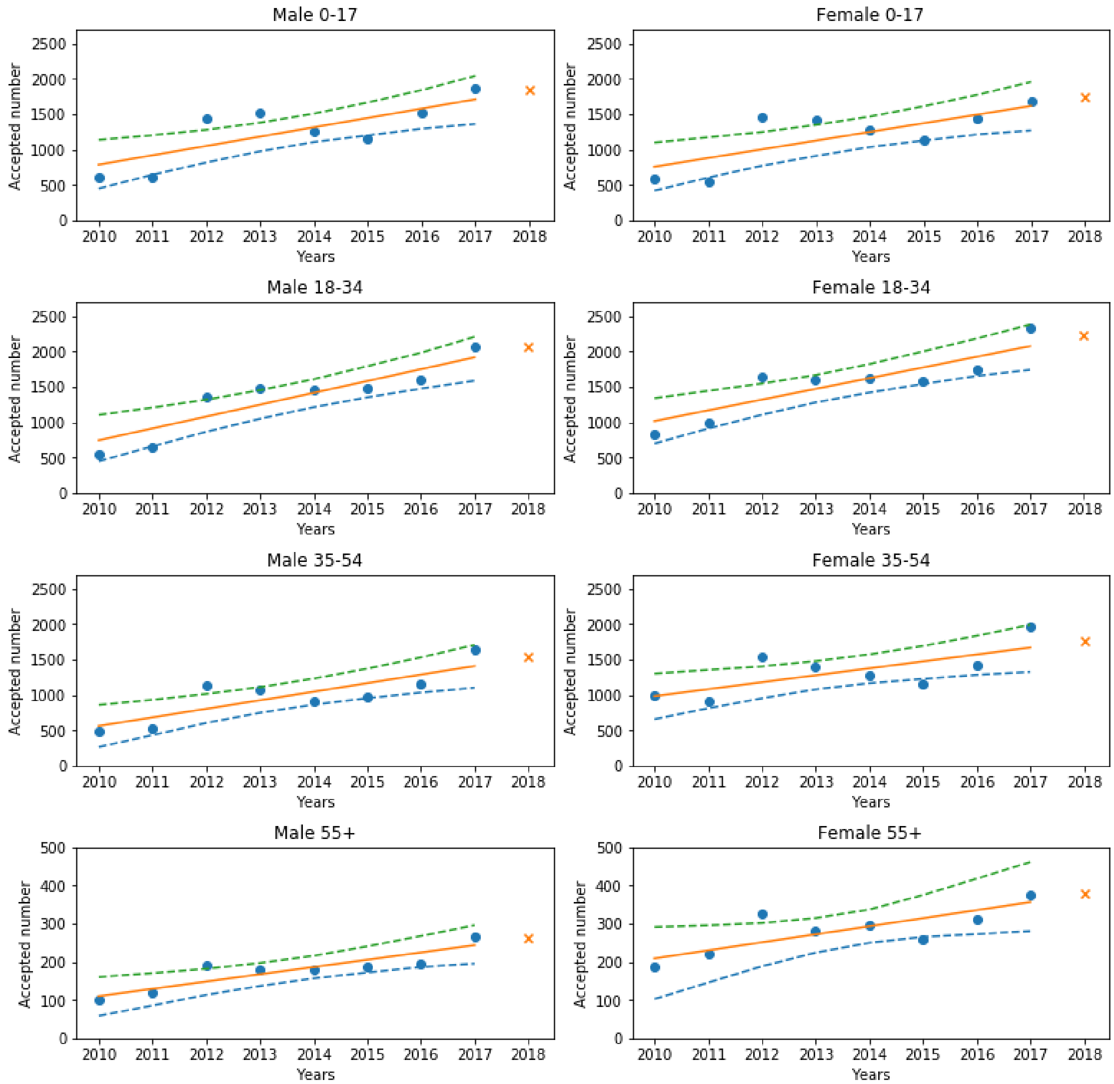
5.6.2 K values and PSIS measurements



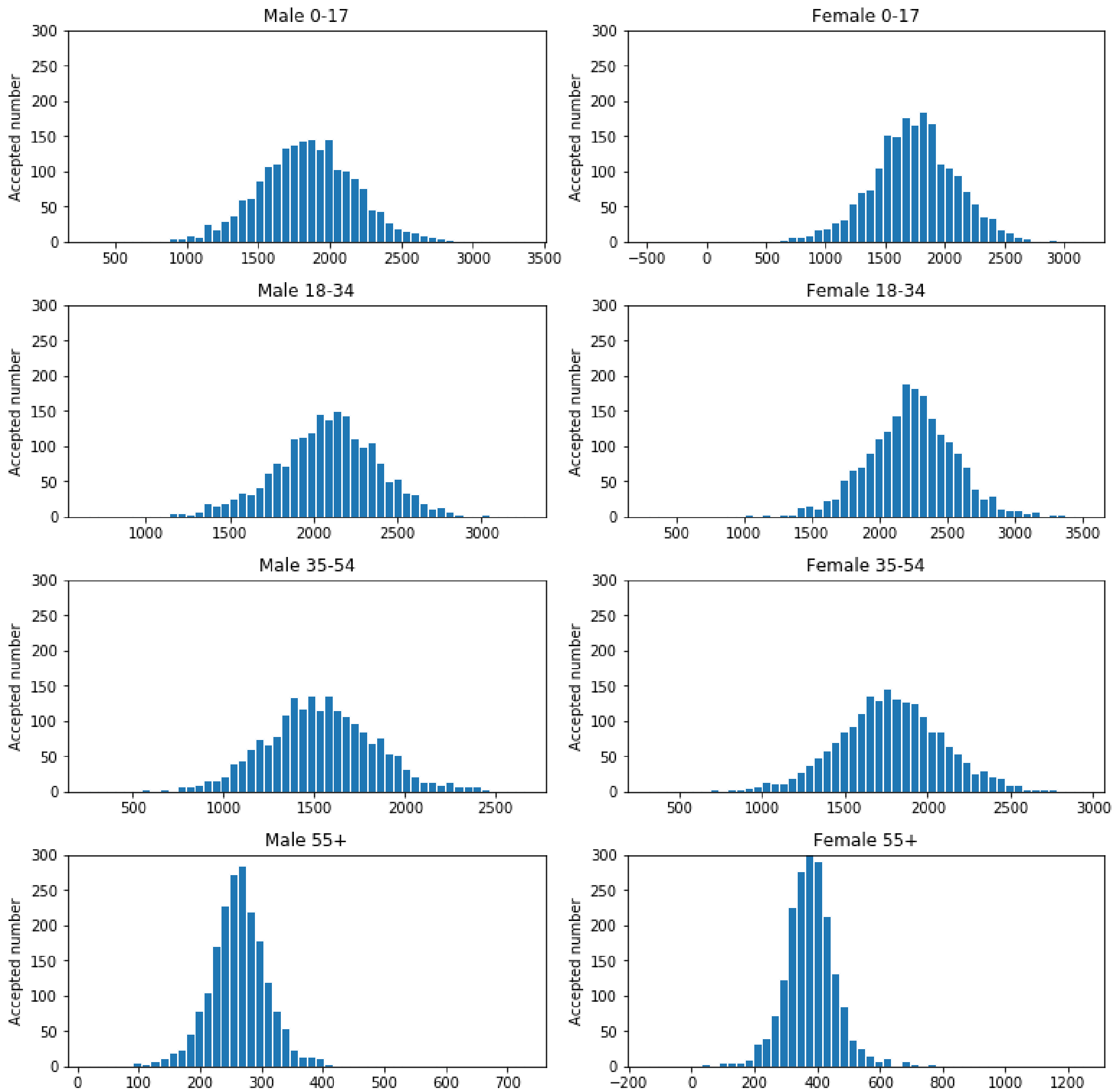
The k values are again decent, and the Psis and Peff values are, respectively -426.7028 and 16.9328. This is beginning to show a slight improvement from the flat prior. Again, the same plots for the mean slope and prediction sampling will be shown below and we will try one last prior, with a modified μ.

5.6.3 Predicting results for the year 2018

We can see a slight improvement from the previous model.



5.6.4 Post Predictive Analysis



Analysis: Same as the previous Model, the model seemed to have performed better on the elder groups and shows better result than the Flat Prior, therefore this model is our best model untill now.

5.7 Hierarchical model for α and σ and normal, more informative prior on mu0_alpha, with an intuition that the slope becomes steeper

The final prior that we will try out has a higher μ value and so, in turn, the slopes - difference between two consecutive years of the number of accepted citizenships - will be loosely restricted to a higher value:

$$\mu_{0alpha} \sim N(200, 300)$$

In this model, we give sigma a higher value, so that it will not enforce our hyperparameter too much, since this prior is only a guess on what should happen on the next years, based subjective information (for instance, increasing immigrations due to Middle East problems etc).

Please find the Stand Code for the model: Link to model

5.7.1 Convergence analysis

	alpha[0]	alpha[1]	alpha[2]	alpha[3]	alpha[4]	alpha[5]	alpha[6]	alpha[7]
n_eff	1434.0	1587.0	1488.0	544.0	1519.00	1209.0	1477.0	299.0
Rhat	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.01

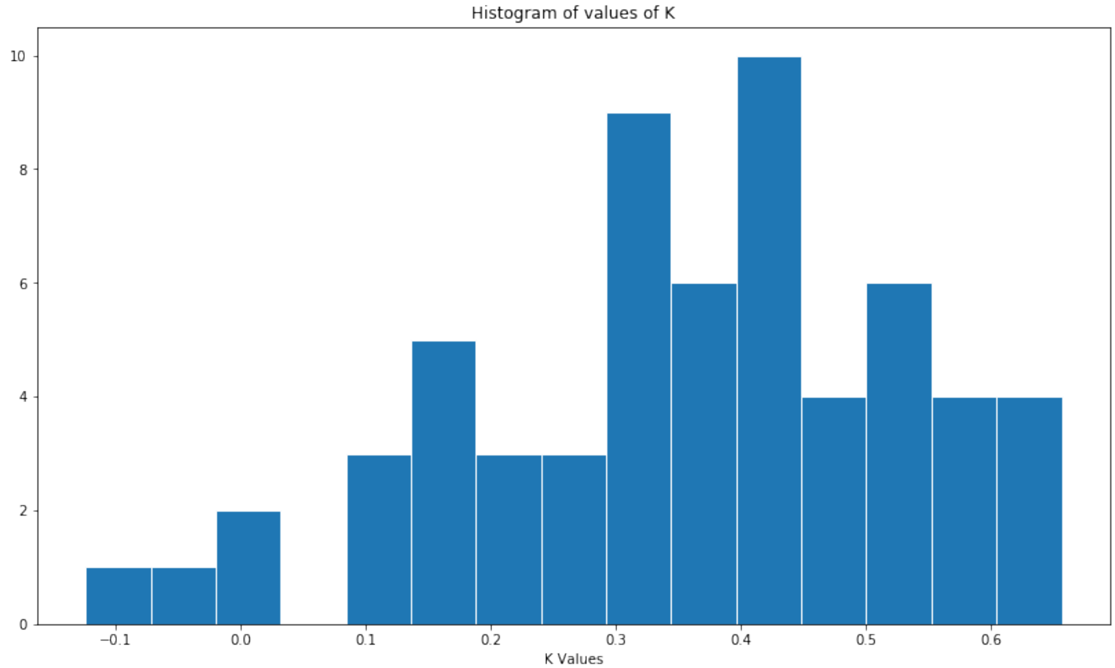
	beta[0]	beta[1]	beta[2]	beta[3]	beta[4]	beta[5]	beta[6]	beta[7]
n_eff	1434.0	1587.0	1488.0	544.0	1519.00	1209.0	1477.0	299.0
Rhat	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.01

	sigma[0]	sigma[1]	sigma[2]	sigma[3]	sigma[4]	sigma[5]	sigma[6]	sigma[7]
n_eff	1129.00	2000.0	1473.0	381.00	1183.0	1666.0	1204.0	331.0
Rhat	1.0	1.0	1.0	1.02	1.0	1.0	1.0	1.02

	mu0_alpha	sigma0_alpha	p1	p2
n_eff	1341.0	918.0	1497.0	623.0
Rhat	1.0	1.0	1.0	1.01

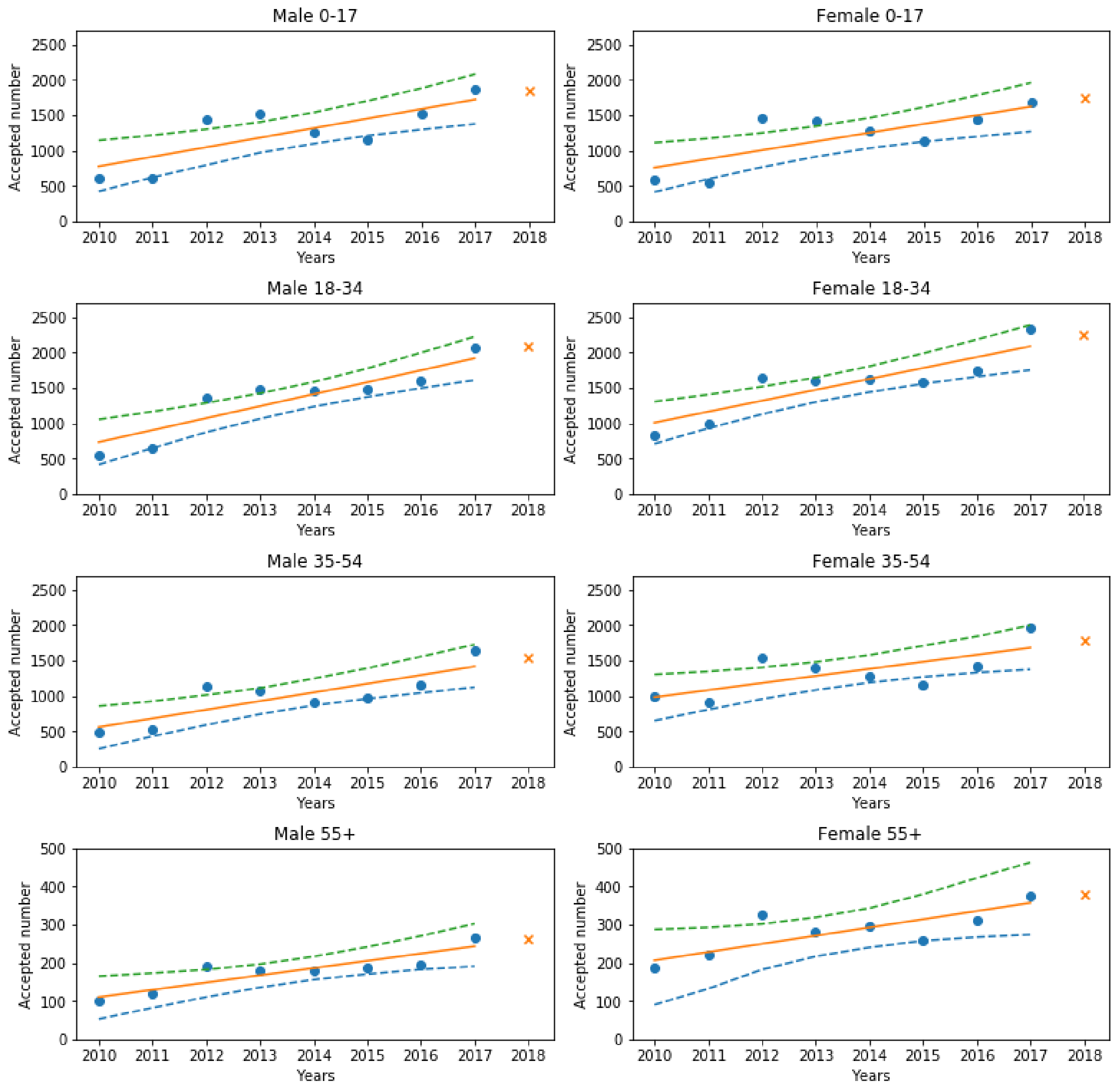
Again, models look good and they have good convergence values of $\hat{R}$ and n_{eff}.

5.7.2 K values and PSIS measurements



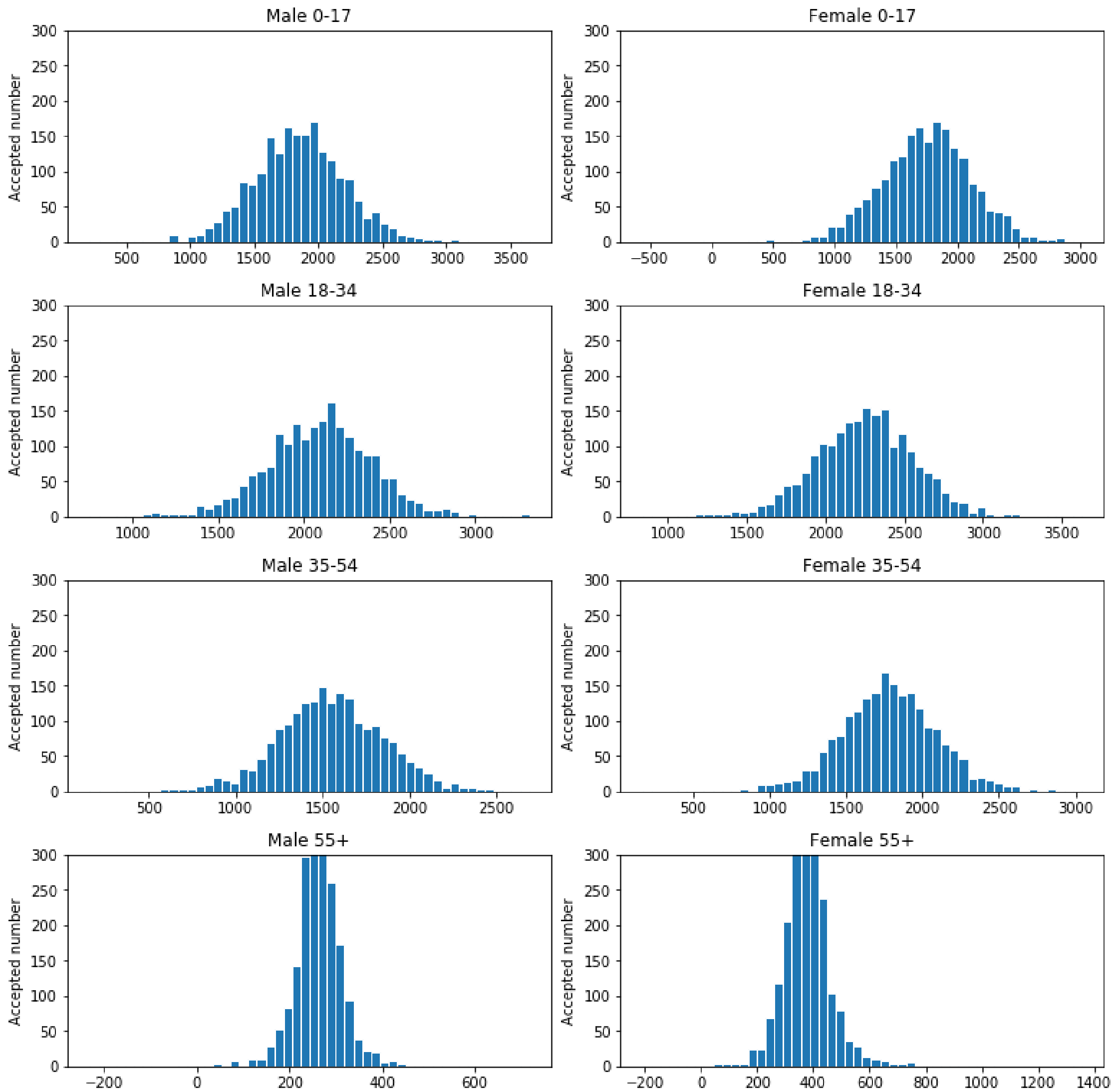
Psis value for this model, -427.2411 and Peff 17.3903 are just a tiny bit better then this model with no priors (flat prior). Although the intuition that the slope should be increasing over the years seems to be good, it looks like the model still wasn't affected too much by our slightly informative prior.

5.7.3 Predictions for the year 2018



We can only see only slight variations from the previous model.

5.7.4 Post Predictive Analysis



Again, the plots are very similar to the ones we got through the other prior variation for this specific model. Only the variation on the elderly groups may have been further reduced.
Overall, the conclusion would be that this model is better than all the other ones and that it's sensitivity to priors is low and gives consistent results.

6 Conclusion and summary

The results we got are mostly as expected. Visually, the predictions for the year 2018 seems to be reasonable for every model.

None of the models had a strong prior sensitivity, but this may also be because of the priors being weakly informative (the variance of our normal priors taken into consideration was high enough).

The results, summarized in a table:

Model	Maximum $\hat{R}$	Prior	Psis	Peff
Separate	1.01	Flat prior	-429.85	16.99
Separate	1.01	N(21.46,168.07)	-430.30	17.40
Hierarchical α	1.01	Flat prior	-429.31	16.99
Hierarchical β	1.01	Flat prior	-428.78	16.38
Hierarchical α-σ	1.01	Flat prior	-426.91	17.11
Hierarchical α-σ	1.01	N(21.46,400)	-427.19	17.46
Hierarchical α-σ	1.02	N(21.46,168)	-426.70	16.93
Hierarchical α-σ	1.02	N(200,300)	-427.24	17.39

BEST MODEL: After a thorough analysis, we conclude and recommend MIGRI to use the **Hierarchical $\boldsymbol{\alpha}$-$\boldsymbol{\sigma}$** model. It's prior sensitivity isn't noticeable, but a good consideration of the prior, based on the above results, would be $N(21.46, 168)$.

7 Future improvements

Along these models, other interesting things that could be tried out may be a double-level hierarchy model. Instead of treating each age category from each the two genders separately, we may combine them such that one gender will have a hierarchy on four parameters (maybe on slope α, like it was done in this project), and above that, combine the hyperparameters of the groups together on another level.

Also, we tried out a couple of interesting hierarchies that didn't work out properly, like a hierarchy on σ parameters using an inverse-gamma or inverse-chi-squared. This may be interesting to further look into.

STAN AND PYTHON CODE

8 CODE

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.pylab as pylab
        import pystan
        import psis
        from plot_results import plot_results
        import importlib
        from plot_pooled_results import plot_results_pooled
```

```
In [2]: from plot_results import plot_results
```

```
In [3]: data = pd.read_csv("data/data.csv",sep='\t',header=None)
```

```
In [4]: np_data =  data.values
        years = np_data[:,0]
        males = np_data[:,1:5]
        females = np_data[:,5:9]
        all_people = np_data[:,1:]
```

8.1 Description of the data

```
In [5]: header = [np.array(['Male','Male','Male','Male','Female','Female','Female','Female']),
        np.array(['0-17','18-34','35-54','55+','0-17','18-34','35-54','55+'])]
        df = pd.DataFrame(all_people, index=data[0].values, columns = header )
        print(df)
```

8.2 Plot data with different colours for each category

```
In [6]: from pylab import rcParams
        import matplotlib.patches as mpatches
        years = data[0].values
        colors = ['red','blue','green','orange','red','blue','green','orange']
        marks = ['s','s','s','s','^','^','^','^']
        legend_labels = ['Male 0-17','Male 18-34','Male 35-54','Male 55+',
                         'Female 0-17','Female 18-34','Female 35-54','Female 55+']
        rcParams['figure.figsize'] = 14, 8
        patches = []
        for i in range(8):
            patches.append(mpatches.Patch(color=colors[i],label=legend_labels[i]))
```

```
In [7]: #plt.legend(handles=patches, loc=0, borderaxespad=0)
        plt.xlabel("Year")
        plt.ylabel("Number of people")
        for ind, gender, category in (zip(range(0,8),header[0],header[1])):
            plt.scatter(years,df[:][gender][category].values,c=colors[ind],
                        marker=marks[ind],s=70,label=legend_labels[ind])
```

```
plt.legend()
plt.show()
```

```
In [8]: means = np.sum(all_people,axis=1)
        means = means/8
        pylab.plot(range(2010,2018),means)
        z = np.polyfit(range(2010,2018), means, 1)
        p = np.poly1d(z)
        plt.xlabel("Years")
        plt.ylabel("Number of accepted citizens")
        pylab.plot(range(2010,2018),p(range(2010,2018)),"r--")
```

8.3 Separate model with flat prior

```
In [9]: population_model_seaprate ="""
        data{
            int<lower=0> N;          // number of data points
            int<lower=0> G;          // number of groups
            vector[N] years;         // years 2010...2017
            vector<lower=0>[N] y;    // the actual number of accepted citizens
            int grps_ind[N];         // group indicator vector (1,2,...,8,1,2...,8...)
            real xpred;              // year 2018
        }
        parameters{
            vector[G] alpha;            // regression paramters for our
            vector[G] beta;             // data points means (separate for each age category)
            vector<lower=0>[G] sigma;   // common sigma for each group
        }
        transformed parameters{
            vector[N] mu;
            for (i in 1:N)
                mu[i] =  alpha[grps_ind[i]]*years[i] + beta[grps_ind[i]];
                                                        // transformed mu as a
                                                        // linear function of years
        }
        model{
            for (i in 1:N)
                y[i] ~ normal(mu[i],sigma[grps_ind[i]]);
                                    // normal distribution around
                                    // each point
        }
        generated quantities {
            vector[G] y_pred_18;          //predictions for each age category
            vector[N] log_lik;
            real x_pred;
            for (i in 1:G)
                y_pred_18[i] = normal_rng(alpha[grps_ind[i]]*xpred +
                                    beta[grps_ind[i]],sigma[grps_ind[i]]);
```

```
            for (i in 1:N)
                log_lik[i] = normal_lpdf(y[i] | mu[i], sigma[grps_ind[i]]);
                                                    //log likelyhood values
        }

        """
        separate = pystan.StanModel(model_code=population_model_seaprate)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_ebec5450bc9a61212360b4ef1ebb9714 NOW.
```

```
In [10]: years = np.array(list(range(2010,2018,1))).reshape((8,1))
         years = np.tile(years,8).flatten()

         separate_data ={
             'N': all_people.size,
             'G':8,
             'years':years,
             'y': all_people.flatten(),
             'grps_ind': list(range(1,9,1))*8,
             'xpred': 2018,
         }
```

```
In [17]: fit_seaprate_no_prior = separate.sampling(data=separate_data,
                      iter=1000,chains=4,control={"max_treedepth":20})
```

```
In [186]: summary = fit_seaprate_no_prior.summary(pars=['alpha','beta','sigma'])

          summary = fit_seaprate_no_prior.summary(pars=['alpha','beta','sigma'])

          df = pd.DataFrame(
              summary['summary'],
              index = summary['summary_rownames'],
              columns = summary['summary_colnames'])
```

```
In [187]: df.loc['alpha[0]':'alpha[9]','n_eff':'Rhat'].T.round(2)
```

Out[187]:

	alpha[0]	alpha[1]	alpha[2]	alpha[3]	alpha[4]	alpha[5]	alpha[6]	\
n_eff	692.0	510.0	657.0	552.00	1098.0	849.00	854.0	
Rhat	1.0	1.0	1.0	1.01	1.0	1.01	1.0	

	alpha[7]
n_eff	682.0
Rhat	1.0

```
In [188]: df.loc['beta[0]':'beta[8]','n_eff':'Rhat'].T.round(2)
```

Out[188]:

	beta[0]	beta[1]	beta[2]	beta[3]	beta[4]	beta[5]	beta[6]	beta[7]
n_eff	692.0	510.0	657.0	552.00	1098.0	849.00	854.0	682.0
Rhat	1.0	1.0	1.0	1.01	1.0	1.01	1.0	1.0

In [189]: df.loc['sigma[0]':'sigma[8]','n_eff':'Rhat'].T.round(2)

```
Out[189]:        sigma[0]  sigma[1]  sigma[2]  sigma[3]  sigma[4]  sigma[5]  sigma[6]  \
          n_eff     760.0     677.0     710.0    862.00     985.0     775.0     797.0
          Rhat        1.0       1.0       1.0      1.01       1.0       1.0       1.0

                 sigma[7]
          n_eff     764.0
          Rhat        1.0
```

K values and log psis measurements

```
In [231]: results = psis.psisloo(fit_seaprate_no_prior.extract()['log_lik'])
          log_lik = fit_seaprate_no_prior.extract()['log_lik']
          means = np.mean(np.exp(log_lik),axis=0)
          sums = np.sum(np.log(means))
          p_eff = sums  - results[0]
          k_vals=psis.psisloo(fit_seaprate_no_prior.extract()['log_lik'])[2]
          plt.hist(k_vals,bins=15, ec='white')
          plt.title("Histogram of values of K")
          plt.xlabel("K Values")
          plt.show()
```

```
In [23]: print(results[0])
```

```
-429.85609031128047
```

```
In [24]: print(p_eff)
```

```
16.994522836727015
```

```
In [25]: summary = fit_seaprate_no_prior.summary(pars=['mu'])
```

```
In [26]: df = pd.DataFrame(
             summary['summary'],
             index = summary['summary_rownames'],
             columns = summary['summary_colnames'])
```

```
In [29]: all_mus2_5 = []
         all_mus97_5 = []
         for i in range(0,64):
             current_mu = df.loc['mu[' + str(i)+ "]"]
             all_mus2_5.append(current_mu[3])
             all_mus97_5.append(current_mu[7])
```

```
In [30]: all_mus2_5 = np.array(all_mus2_5)
         all_mus97_5 = np.array(all_mus97_5)
         all_mus2_5 = np.reshape(all_mus2_5,(8,8))
         all_mus97_5 = np.reshape(all_mus97_5,(8,8))
```

```
In [31]: plot_results(fit_seaprate_no_prior,all_mus2_5,all_mus97_5,all_people)

In [33]: new_predictions = fit_seaprate_no_prior.extract()['y_pred_18']
         new_predictions = new_predictions.T
         def set_size(w,h, ax=None):
                 """ w, h: width, height in inches """
                 if not ax: ax=plt.gca()
                 l = ax.figure.subplotpars.left
                 r = ax.figure.subplotpars.right
                 t = ax.figure.subplotpars.top
                 b = ax.figure.subplotpars.bottom
                 figw = float(w)/(r-l)
                 figh = float(h)/(t-b)
                 ax.figure.set_size_inches(figw, figh)

         import matplotlib.gridspec as gridspec
         import matplotlib.pyplot as plt
         from matplotlib import figure
         gs = gridspec.GridSpec(4,6)
         plot_titles = ["Male 0-17","Male 18-34","Male 35-54","Male 55+",
                        "Female 0-17","Female 18-34","Female 35-54","Female 55+"]
         plot_indexes = np.array([[0,3],[3,6],[0,3],[3,6],[0,3],[3,6],[0,3],[3,6]])

         for i in range(4):
             plt.tight_layout(pad=0.2, w_pad=0.5, h_pad=1.0)

             ax = plt.subplot(gs[i,0:3])
             set_size(10,10)
             ax.set_ylabel("Accepted number")
             ax.set_title(plot_titles[i])
             ax.set_ylim(0,300)
             ax.hist(new_predictions[i],bins=50,ec='white')
             plt.tight_layout(pad=0.2, w_pad=0.5, h_pad=1.0)

             ax = plt.subplot(gs[i,3:6])
             set_size(10,10)
             ax.set_ylabel("Accepted number")
             ax.set_title(plot_titles[i+4])
             ax.set_ylim(0,300)
             ax.hist(new_predictions[i+4],bins=50,ec='white')
         plt.show()

In [34]: total_citizens_1990 = pd.read_csv('data/total_accepted.csv',header=None)
         plt.ylabel("Number of accepted citizenships")
         plt.xlabel("Year")
         plt.plot(total_citizens_1990.values[:,0],total_citizens_1990.values[:,1])

Out[34]: [<matplotlib.lines.Line2D at 0x2a236262550>]
```

```
In [35]: people_2009 = total_citizens_1990
         people_2009 = people_2009.values[:,1]
         diff_2009 = people_2009[1:21] - people_2009[:20]

         avg_diff_2009 = diff_2009/8

         print(np.mean(avg_diff_2009))
         print(np.std(avg_diff_2009))
```

```
21.46875
168.07122687922373
```

8.4 Separate model with normal prior for year parameter

```
In [36]: population_model_seaprate ="""
         data{
             real prior_sigma;
             real prior_mu;
             int<lower=0> N;          // number of data points
             int<lower=0> G;          // number of groups
             vector[N] years;         // years 2010...2017
             vector<lower=0>[N] y;    // the actual number of accepted citizens
             int grps_ind[N];         // group indicator vector (1,2,...,8,1,2...,8...)
             real xpred;              // year 2018
         }
         parameters{
             vector[G] alpha;            // regression paramters for our
             vector[G] beta;             // data points means (separate for each age category)
             vector<lower=0>[G] sigma;   // common sigma for each group
         }
         transformed parameters{
             vector[N] mu;
             for (i in 1:N)
                 mu[i] =  alpha[grps_ind[i]]*years[i] + beta[grps_ind[i]];
                                                 // transformed mu as a
                                                 // linear function of years
         }
         model{
             alpha ~ normal(prior_mu,prior_sigma);
                     // prior for the alpha parameter
             for (i in 1:N)
                 y[i] ~ normal(mu[i],sigma[grps_ind[i]]);
                             // normal distribution around
                             // each point
         }
         generated quantities {
             vector[G] y_pred_18;
```

```
        vector[N] log_lik;
        real x_pred;
        for (i in 1:G)
            y_pred_18[i] = normal_rng(alpha[grps_ind[i]]*xpred +
                                beta[grps_ind[i]],sigma[grps_ind[i]]);

        for (i in 1:N)
            log_lik[i] = normal_lpdf(y[i] | mu[i], sigma[grps_ind[i]]);

    }

    """
    separate_with_prior = pystan.StanModel(model_code=population_model_seaprate)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_4c9a74783aab0891ae1608cc2393825e NOW.
```

In [37]:
```
years = np.array(list(range(2010,2018,1))).reshape((8,1))
years = np.tile(years,8).flatten()

separate_data ={
    'N': all_people.size,
    'G':8,
    'years':years,
    'y': all_people.flatten(),
    'grps_ind': list(range(1,9,1))*8,
    'xpred': 2018,
    'prior_sigma': 168.071,
    'prior_mu': 21.46875
}
```

In [38]:
```
fit_separate_prior = separate_with_prior.sampling(data=separate_data,iter=1000,chains=4
```

```
D:\Anaconda3\lib\site-packages\pystan\misc.py:399: FutureWarning: Conversion of the second argum
  elif np.issubdtype(np.asarray(v).dtype, float):
```

In [190]:
```
summary = fit_separate_prior.summary(pars=['alpha','beta','sigma'])

df = pd.DataFrame(
    summary['summary'],
    index = summary['summary_rownames'],
    columns = summary['summary_colnames']).round(2)

df.loc['alpha[0]':'alpha[8]','n_eff':'Rhat'].T.round(2)
```

Out[190]:

	alpha[0]	alpha[1]	alpha[2]	alpha[3]	alpha[4]	alpha[5]	alpha[6]	\
n_eff	744.0	613.0	1090.0	600.00	807.00	961.0	892.0	
Rhat	1.0	1.0	1.0	1.01	1.01	1.0	1.0	

```
         alpha[7]
n_eff       871.0
Rhat          1.0
```

In [193]:
```
df.loc['beta[0]':'beta[8]','n_eff':'Rhat'].T
```

Out[193]:
```
         beta[0]  beta[1]  beta[2]  beta[3]  beta[4]  beta[5]  beta[6]  beta[7]
n_eff     743.0    613.0   1090.0   600.00   807.00    961.0    892.0    871.0
Rhat        1.0      1.0      1.0     1.01     1.01      1.0      1.0      1.0
```

In [192]:
```
df.loc['sigma[0]':'sigma[8]','n_eff':'Rhat'].T
```

Out[192]:
```
         sigma[0]  sigma[1]  sigma[2]  sigma[3]  sigma[4]  sigma[5]  sigma[6]  \
n_eff     428.00     729.0    1003.0     699.0     708.0    1206.0    847.00
Rhat        1.01       1.0       1.0       1.0       1.0       1.0      1.01

         sigma[7]
n_eff       947.0
Rhat          1.0
```

K values and log psis measurements

In [232]:
```
results = psis.psisloo(fit_separate_prior.extract()['log_lik'])
log_lik = fit_separate_prior.extract()['log_lik']
means = np.mean(np.exp(log_lik),axis=0)
sums = np.sum(np.log(means))
p_eff = sums  - results[0]
plt.hist(k_vals,bins=15,ec='white')
plt.title("Histogram of values of K")
plt.xlabel("K Values")
plt.show()
```

In [55]:
```
print(results[0])
```

```
-430.30148743967766
```

In [56]:
```
print(p_eff)
```

```
17.403591739994567
```

In [57]:
```
summary = fit_separate_prior.summary(pars=['mu'])
```

In [58]:
```
df = pd.DataFrame(
    summary['summary'],
    index = summary['summary_rownames'],
    columns = summary['summary_colnames'])
```

In [62]:
```
all_mus2_5 = []
all_mus97_5 = []
for i in range(0,64):
    current_mu = df.loc['mu[' + str(i)+ "]"]
    all_mus2_5.append(current_mu[3])
    all_mus97_5.append(current_mu[7])
```

In [63]:
```
all_mus2_5 = np.array(all_mus2_5)
all_mus97_5 = np.array(all_mus97_5)
all_mus2_5 = np.reshape(all_mus2_5,(8,8))
all_mus97_5 = np.reshape(all_mus97_5,(8,8))
```

In [64]:
```
plot_results(fit_separate_prior,all_mus2_5,all_mus97_5,all_people)
```

In [65]:
```
new_predictions = fit_separate_prior.extract()['y_pred_18']
new_predictions = new_predictions.T
def set_size(w,h, ax=None):
        """ w, h: width, height in inches """
        if not ax: ax=plt.gca()
        l = ax.figure.subplotpars.left
        r = ax.figure.subplotpars.right
        t = ax.figure.subplotpars.top
        b = ax.figure.subplotpars.bottom
        figw = float(w)/(r-l)
        figh = float(h)/(t-b)
        ax.figure.set_size_inches(figw, figh)

import matplotlib.gridspec as gridspec
import matplotlib.pyplot as plt
from matplotlib import figure
gs = gridspec.GridSpec(4,6)
plot_titles = ["Male 0-17","Male 18-34","Male 35-54","Male 55+",
                      "Female 0-17","Female 18-34","Female 35-54","Female 55+"]
plot_indexes = np.array([[0,3],[3,6],[0,3],[3,6],[0,3],[3,6],[0,3],[3,6]])

for i in range(4):
    plt.tight_layout(pad=0.2, w_pad=0.5, h_pad=1.0)

    ax = plt.subplot(gs[i,0:3])
    set_size(10,10)
    ax.set_ylabel("Accepted number")
    ax.set_title(plot_titles[i])
    ax.set_ylim(0,300)
    ax.hist(new_predictions[i],bins=50,ec='white')
    plt.tight_layout(pad=0.2, w_pad=0.5, h_pad=1.0)

    ax = plt.subplot(gs[i,3:6])
```

```
        set_size(10,10)
        ax.set_ylabel("Accepted number")
        ax.set_title(plot_titles[i+4])
        ax.set_ylim(0,300)
        ax.hist(new_predictions[i+4],bins=50,ec='white')
    plt.show()
```

8.5 Hierarchical model for the slope α

$y_i \propto p(y_i|\mu_i, \sigma_{grp-i})p(\mu|\alpha, \beta)$

```
In [66]: population_model_hierarchical ="""
    data{
        int<lower=0> N;          // number of data points
        int<lower=0> G;          // number of groups
        vector[N] years;         // years 2010...2017
        vector<lower=0>[N] y;    // the actual number of accepted citizens
        int grps_ind[N];         // group indicator vector (1,2,...,8,1,2...,8...)
        real xpred;              // year 2018
    }
    parameters{
        vector[G] alpha;            // regression paramters for our
        vector[G] beta;             // data points means (separate for each age category)
        vector<lower=0>[G] sigma;   // common sigma for each group
        real mu0;
        real<lower=0> sigma0;
    }
    transformed parameters{
        vector[N] mu;
        for (i in 1:N)
            mu[i] =  alpha[grps_ind[i]]*years[i] + beta[grps_ind[i]];
                                                    // transformed mu as a
                                                    // linear function of years
    }
    model{
        alpha ~ normal(mu0,sigma0);
                                    // hierarchy on the slopes
        for (i in 1:N)
            y[i] ~ normal(mu[i],sigma[grps_ind[i]]);
                                    // normal distribution around
                                    // each point;
    }
    generated quantities {
        vector[G] y_pred_18;
        vector[N] log_lik;
        real x_pred;
        for (i in 1:G)
            y_pred_18[i] = normal_rng(alpha[grps_ind[i]]*xpred +
```

```
                             beta[grps_ind[i]],sigma[grps_ind[i]]);

             for (i in 1:N)
                 log_lik[i] = normal_lpdf(y[i] | mu[i], sigma[grps_ind[i]]);

         }

         """
         hierarchical_slope = \
             pystan.StanModel(model_code=population_model_hierarchical)
```

INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_654a7070a34f5b310c2843bdc2ebfc56 NOW.

```
In [67]: years = np.array(list(range(2010,2018,1))).reshape((8,1))
         years = np.tile(years,8).flatten()

         hierarchical_data ={
             'N': all_people.size,
             'G':8,
             'years':years,
             'y': all_people.flatten(),
             'grps_ind': list(range(1,9,1))*8,
             'xpred': 2018
         }
```

```
In [68]: fit_hierarchical_slope = hierarchical_slope.sampling(data=hierarchical_data,
                                     iter=1000,chains=4,control={"max_treedepth":20})
```

```
In [196]: summary = fit_hierarchical_slope.summary(pars=['alpha','beta','sigma','mu0','sigma0'])

          df = pd.DataFrame(
              summary['summary'],
              index = summary['summary_rownames'],
              columns = summary['summary_colnames']).round(2)

          df.loc['alpha[0]':'alpha[7]','n_eff':'Rhat'].T
```

Out[196]:

	alpha[0]	alpha[1]	alpha[2]	alpha[3]	alpha[4]	alpha[5]	alpha[6]	alpha[7]
n_eff	965.00	820.00	631.0	809.0	904.0	617.0	861.0	912.0
Rhat	1.01	1.01	1.0	1.0	1.0	1.0	1.0	1.0

```
In [198]: df.loc['beta[0]':'beta[7]','n_eff':'Rhat'].T
```

Out[198]:

	beta[0]	beta[1]	beta[2]	beta[3]	beta[4]	beta[5]	beta[6]	beta[7]
n_eff	965.00	820.00	631.0	809.0	904.0	617.0	861.0	912.0
Rhat	1.01	1.01	1.0	1.0	1.0	1.0	1.0	1.0

In [200]: df.loc['sigma[0]':'sigma[7]','n_eff':'Rhat'].T

Out[200]:

```
       sigma[0]  sigma[1]  sigma[2]  sigma[3]  sigma[4]  sigma[5]  sigma[6]  \
n_eff    1012.0     823.0     953.0     745.0     861.0    406.00     990.0
Rhat        1.0       1.0       1.0       1.0       1.0      1.01       1.0

       sigma[7]
n_eff    1099.0
Rhat        1.0
```

In [205]: df.loc[['mu0','sigma0'],'n_eff':'Rhat'].T

Out[205]:

```
         mu0  sigma0
n_eff  932.0   942.0
Rhat     1.0     1.0
```

In [233]:

```
results = psis.psisloo(fit_hierarchical_slope.extract()['log_lik'])
log_lik = fit_hierarchical_slope.extract()['log_lik']
means = np.mean(np.exp(log_lik),axis=0)
sums = np.sum(np.log(means))
p_eff = sums  - results[0]
k_vals=psis.psisloo(fit_hierarchical_slope.extract()['log_lik'])[2]
plt.hist(k_vals,bins=15,ec='white')
plt.title("Histogram of values of K")
plt.xlabel("K Values")
plt.show()
```

In [73]: print(results[0])

```
-429.31816917855963
```

In [74]: print(p_eff)

```
16.589212722498132
```

In [75]:

```
summary = fit_hierarchical_slope.summary(pars=['mu'])


df = pd.DataFrame(
    summary['summary'],
    index = summary['summary_rownames'],
    columns = summary['summary_colnames'])

all_mus2_5 = []
all_mus97_5 = []
for i in range(0,64):
    current_mu = df.loc['mu[' + str(i)+ "]"]
```

```
            all_mus2_5.append(current_mu[3])
            all_mus97_5.append(current_mu[7])


        all_mus2_5 = np.array(all_mus2_5)
        all_mus97_5 = np.array(all_mus97_5)
        all_mus2_5 = np.reshape(all_mus2_5,(8,8))
        all_mus97_5 = np.reshape(all_mus97_5,(8,8))
```

```
In [76]: plot_results(fit_hierarchical_slope,all_mus2_5,all_mus97_5,all_people)
```

```
In [77]: new_predictions = fit_hierarchical_slope.extract()['y_pred_18']
         new_predictions = new_predictions.T
         def set_size(w,h, ax=None):
                 """ w, h: width, height in inches """
                 if not ax: ax=plt.gca()
                 l = ax.figure.subplotpars.left
                 r = ax.figure.subplotpars.right
                 t = ax.figure.subplotpars.top
                 b = ax.figure.subplotpars.bottom
                 figw = float(w)/(r-l)
                 figh = float(h)/(t-b)
                 ax.figure.set_size_inches(figw, figh)

         import matplotlib.gridspec as gridspec
         import matplotlib.pyplot as plt
         from matplotlib import figure
         gs = gridspec.GridSpec(4,6)
         plot_titles = ["Male 0-17","Male 18-34","Male 35-54","Male 55+",
                               "Female 0-17","Female 18-34","Female 35-54","Female 55+"]
         plot_indexes = np.array([[0,3],[3,6],[0,3],[3,6],[0,3],[3,6],[0,3],[3,6]])

         for i in range(4):
             plt.tight_layout(pad=0.2, w_pad=0.5, h_pad=1.0)

             ax = plt.subplot(gs[i,0:3])
             set_size(10,10)
             ax.set_ylabel("Accepted number")
             ax.set_title(plot_titles[i])
             ax.set_ylim(0,300)
             ax.hist(new_predictions[i],bins=50,ec='white')
             plt.tight_layout(pad=0.2, w_pad=0.5, h_pad=1.0)

             ax = plt.subplot(gs[i,3:6])
             set_size(10,10)
             ax.set_ylabel("Accepted number")
             ax.set_title(plot_titles[i+4])
             ax.set_ylim(0,300)
```

```
        ax.hist(new_predictions[i+4],bins=50,ec='white')
    plt.show()
```

8.6 Hierarchical model for β

```
In [78]: population_model_hierarchical ="""
    data{
        int<lower=0> N;         // number of data points
        int<lower=0> G;         // number of groups
        vector[N] years;        // years 2010...2017
        vector<lower=0>[N] y;   // the actual number of accepted citizens
        int grps_ind[N];        // group indicator vector (1,2,...,8,1,2...,8...)
        real xpred;             // year 2018
    }
    parameters{
        vector[G] alpha;           // regression paramters for our
        vector[G] beta;            // data points means (separate for each age category)
        vector<lower=0>[G] sigma;  // common sigma for each group
        real mu0_beta;
        real<lower=0>sigma0_beta;
    }
    transformed parameters{
        vector[N] mu;
        for (i in 1:N)
            mu[i] =  alpha[grps_ind[i]]*years[i] + beta[grps_ind[i]];
                                                // transformed mu as a
                                                // linear function of years
    }
    model{
        beta ~ normal(mu0_beta,sigma0_beta);
                        // hierarchy on alpha
        for (i in 1:N)
            y[i] ~ normal(mu[i],sigma[grps_ind[i]]);
                            // normal distribution around
                            // each point;
    }
    generated quantities {
        vector[G] y_pred_18;
        vector[N] log_lik;
        real x_pred;
        for (i in 1:G)
            y_pred_18[i] = normal_rng(alpha[grps_ind[i]]*xpred +
                            beta[grps_ind[i]],sigma[grps_ind[i]]);

        for (i in 1:N)
            log_lik[i] = normal_lpdf(y[i] | mu[i], sigma[grps_ind[i]]);

    }
```

```
"""
hierarchical_beta = \
    pystan.StanModel(model_code=population_model_hierarchical)
```

INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_859fde05b9d40f754ed8fa3c80d70c3a NOW.

In [79]:
```
years = np.array(list(range(2010,2018,1))).reshape((8,1))
years = np.tile(years,8).flatten()

hierarchical_data ={
    'N': all_people.size,
    'G':8,
    'years':years,
    'y': all_people.flatten(),
    'grps_ind': list(range(1,9,1))*8,
    'xpred': 2018
}
```

In [81]:
```
fit_hierarchical_beta = hierarchical_beta.sampling(data=hierarchical_data,
        iter=1000,chains=4,control={"max_treedepth":20,"adapt_delta":0.95})
```

In [208]:
```
summary = fit_hierarchical_beta.summary(pars=['alpha','beta','sigma','mu0_beta','sigma

 df = pd.DataFrame(
     summary['summary'],
     index = summary['summary_rownames'],
     columns = summary['summary_colnames']).round(2)

 df.loc['alpha[0]':'alpha[7]','n_eff':'Rhat'].T
```

Out[208]:

	alpha[0]	alpha[1]	alpha[2]	alpha[3]	alpha[4]	alpha[5]	alpha[6]	\
n_eff	1162.0	668.00	1067.0	789.00	1050.0	1275.0	1256.0	
Rhat	1.0	1.01	1.0	1.01	1.0	1.0	1.0	

	alpha[7]
n_eff	1058.0
Rhat	1.0

In [83]:
```
df.loc['beta[0]':'beta[7]','n_eff':'Rhat'].T
```

Out[83]:

	beta[0]	beta[1]	beta[2]	beta[3]	beta[4]	\
n_eff	1163.000000	668.000000	1067.000000	789.000000	1050.000000	
Rhat	1.000676	1.005865	1.003595	1.005063	1.003833	

	beta[5]	beta[6]	beta[7]
n_eff	1276.000000	1256.000000	1058.000000
Rhat	0.999821	0.999675	1.004866

```
In [84]: df.loc['sigma[0]':'sigma[7]','n_eff':'Rhat'].T
```

```
Out[84]:           sigma[0]    sigma[1]     sigma[2]    sigma[3]    sigma[4]  \
         n_eff  1127.000000  853.000000  1163.000000  894.000000  977.000000
         Rhat      1.000613    1.002098     0.999276    1.001141    1.001863

                  sigma[5]     sigma[6]     sigma[7]
         n_eff  1059.00000  1319.000000  1082.000000
         Rhat      1.00331     1.000156     1.002458
```

```
In [210]: df.loc[['mu0_beta','sigma0_beta'],'n_eff':'Rhat'].T
```

```
Out[210]:        mu0_beta  sigma0_beta
          n_eff    1436.0        967.0
          Rhat        1.0          1.0
```

```
In [234]: results = psis.psisloo(fit_hierarchical_beta.extract()['log_lik'])
          log_lik = fit_hierarchical_beta.extract()['log_lik']
          means = np.mean(np.exp(log_lik),axis=0)
          sums = np.sum(np.log(means))
          p_eff = sums  - results[0]
          k_vals = psis.psisloo(fit_hierarchical_beta.extract()['log_lik'])[2]
          plt.hist(k_vals,bins=15,ec='white')
          plt.title("Histogram of values of K")
          plt.xlabel("K Values")
          plt.show()
```

```
In [101]: print(results[0])
```

```
-428.7834910679668
```

```
In [102]: print(p_eff)
```

```
16.38814096350154
```

```
In [103]: summary = fit_hierarchical_beta.summary(pars=['mu'])


          df = pd.DataFrame(
              summary['summary'],
              index = summary['summary_rownames'],
              columns = summary['summary_colnames'])

          all_mus2_5 = []
          all_mus97_5 = []
          for i in range(0,64):
              current_mu = df.loc['mu[' + str(i)+ "]"]
```

```
        all_mus2_5.append(current_mu[3])
        all_mus97_5.append(current_mu[7])


    all_mus2_5 = np.array(all_mus2_5)
    all_mus97_5 = np.array(all_mus97_5)
    all_mus2_5 = np.reshape(all_mus2_5,(8,8))
    all_mus97_5 = np.reshape(all_mus97_5,(8,8))
```

In [104]:
```
plot_results(fit_hierarchical_beta,all_mus2_5,all_mus97_5,all_people)
```

In [105]:
```
new_predictions = fit_hierarchical_beta.extract()['y_pred_18']
new_predictions = new_predictions.T
def set_size(w,h, ax=None):
        """ w, h: width, height in inches """
        if not ax: ax=plt.gca()
        l = ax.figure.subplotpars.left
        r = ax.figure.subplotpars.right
        t = ax.figure.subplotpars.top
        b = ax.figure.subplotpars.bottom
        figw = float(w)/(r-l)
        figh = float(h)/(t-b)
        ax.figure.set_size_inches(figw, figh)

import matplotlib.gridspec as gridspec
import matplotlib.pyplot as plt
from matplotlib import figure
gs = gridspec.GridSpec(4,6)
plot_titles = ["Male 0-17","Male 18-34","Male 35-54","Male 55+",
                      "Female 0-17","Female 18-34","Female 35-54","Female 55+"]
plot_indexes = np.array([[0,3],[3,6],[0,3],[3,6],[0,3],[3,6],[0,3],[3,6]])

for i in range(4):
    plt.tight_layout(pad=0.2, w_pad=0.5, h_pad=1.0)

    ax = plt.subplot(gs[i,0:3])
    set_size(10,10)
    ax.set_ylabel("Accepted number")
    ax.set_title(plot_titles[i])
    ax.set_ylim(0,300)
    ax.hist(new_predictions[i],bins=50,ec='white')
    plt.tight_layout(pad=0.2, w_pad=0.5, h_pad=1.0)

    ax = plt.subplot(gs[i,3:6])
    set_size(10,10)
    ax.set_ylabel("Accepted number")
    ax.set_title(plot_titles[i+4])
    ax.set_ylim(0,300)
```

```
        ax.hist(new_predictions[i+4],bins=50,ec='white')
    plt.show()
```

8.7 Hierarchical model for α and σ and flat prior

```
In [106]: population_model_hierarchical ="""
    data{
        int<lower=0> N;         // number of data points
        int<lower=0> G;         // number of groups
        vector[N] years;        // years 2010...2017
        vector<lower=0>[N] y;   // the actual number of accepted citizens
        int grps_ind[N];        // group indicator vector (1,2,...,8,1,2...,8...)
        real xpred;             // year 2018
    }
    parameters{
        vector[G] alpha;           // regression paramters for our
        vector[G] beta;            // data points means (separate for each age category)
        vector<lower=0>[G] sigma;  // common sigma for each group
        real mu0_alpha;
        real<lower=0>sigma0_alpha;
        real p1;//parameters for cauchy
        real p2;//parameters for cauchy
    }
    transformed parameters{
        vector[N] mu;
        for (i in 1:N)
            mu[i] =  alpha[grps_ind[i]]*years[i] + beta[grps_ind[i]];
                                                  // transformed mu as a
                                                  // linear function of years
    }
    model{
        sigma ~ cauchy(p1, p2);
                            // hierarchy on sigma
        alpha ~ normal(mu0_alpha,sigma0_alpha);
                            // hierarchy on alpha
        for (i in 1:N)
            y[i] ~ normal(mu[i],sigma[grps_ind[i]]);
                        // normal distribution around
                        // each point;

    }
    generated quantities {
        vector[G] y_pred_18;
        vector[N] log_lik;
        real x_pred;
        for (i in 1:G)
            y_pred_18[i] = normal_rng(alpha[grps_ind[i]]*xpred +
                            beta[grps_ind[i]],sigma[grps_ind[i]]);
```

```
    for (i in 1:N)
        log_lik[i] = normal_lpdf(y[i] | mu[i], sigma[grps_ind[i]]);

}

"""
hierarchical_both_parameters = \
    pystan.StanModel(model_code=population_model_hierarchical)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_73a79f0f691e7fed00bdd710abaab9ba NOW.
```

```
In [107]: years = np.array(list(range(2010,2018,1))).reshape((8,1))
          years = np.tile(years,8).flatten()

          hierarchical_data ={
              'N': all_people.size,
              'G':8,
              'years':years,
              'y': all_people.flatten(),
              'grps_ind': list(range(1,9,1))*8,
              'xpred': 2018
          }
```

```
In [108]: fit_hier_sgm_alpha = hierarchical_both_parameters.sampling(data=hierarchical_data,
                         iter=1000,chains=4,control={"adapt_delta":0.95,"max_treedepth":18})
```

```
In [212]: summary = fit_hier_sgm_alpha.summary(pars=['alpha','beta','sigma','mu0_alpha','sigma0_

          df = pd.DataFrame(
              summary['summary'],
              index = summary['summary_rownames'],
              columns = summary['summary_colnames']).round(2)

          df.loc['alpha[0]':'alpha[7]','n_eff':'Rhat'].T
```

```
Out[212]:        alpha[0]  alpha[1]  alpha[2]  alpha[3]  alpha[4]  alpha[5]  alpha[6]  \
          n_eff    1100.0    1466.0    1423.0    1138.0    1595.0    1429.0    1370.0
          Rhat        1.0       1.0       1.0       1.0       1.0       1.0       1.0

                 alpha[7]
          n_eff    255.00
          Rhat       1.01
```

```
In [213]: df.loc['beta[0]':'beta[7]','n_eff':'Rhat'].T
```

```
Out[213]:        beta[0]  beta[1]  beta[2]  beta[3]  beta[4]  beta[5]  beta[6]  beta[7]
          n_eff   1100.0   1465.0   1423.0   1138.0   1595.0   1429.0   1371.0   255.00
          Rhat       1.0      1.0      1.0      1.0      1.0      1.0      1.0     1.01
```

In [214]: df.loc['sigma[0]':'sigma[7]','n_eff':'Rhat'].T

Out[214]:

```
       sigma[0]  sigma[1]  sigma[2]  sigma[3]  sigma[4]  sigma[5]  sigma[6]  \
n_eff    1193.0    1349.0    1157.0    740.00    1102.0    1296.0    1457.0
Rhat        1.0       1.0       1.0      1.01       1.0       1.0       1.0

       sigma[7]
n_eff    258.00
Rhat       1.01
```

In [217]: df.loc[['mu0_alpha','sigma0_alpha','p1','p2'],'n_eff':'Rhat'].T

Out[217]:

```
       mu0_alpha  sigma0_alpha      p1      p2
n_eff     1261.0         967.0  1317.0  368.00
Rhat         1.0           1.0     1.0    1.01
```

In [235]:

```
results = psis.psisloo(fit_hier_sgm_alpha.extract()['log_lik'])
log_lik = fit_hier_sgm_alpha.extract()['log_lik']
means = np.mean(np.exp(log_lik),axis=0)
sums = np.sum(np.log(means))
p_eff = sums  - results[0]
k_vals=psis.psisloo(fit_hier_sgm_alpha.extract()['log_lik'])[2]
plt.hist(k_vals,bins=15,ec='white')
plt.title("Histogram of values of K")
plt.xlabel("K Values")
plt.show()
```

In [114]: print(results[0])

```
-426.91016543634527
```

In [115]: print(p_eff)

```
17.11810867229235
```

In [116]:

```
summary = fit_hier_sgm_alpha.summary(pars=['mu'])


df = pd.DataFrame(
    summary['summary'],
    index = summary['summary_rownames'],
    columns = summary['summary_colnames'])

all_mus2_5 = []
all_mus97_5 = []
for i in range(0,64):
    current_mu = df.loc['mu[' + str(i)+ "]"]
```

```
            all_mus2_5.append(current_mu[3])
            all_mus97_5.append(current_mu[7])


        all_mus2_5 = np.array(all_mus2_5)
        all_mus97_5 = np.array(all_mus97_5)
        all_mus2_5 = np.reshape(all_mus2_5,(8,8))
        all_mus97_5 = np.reshape(all_mus97_5,(8,8))
```

In [117]:
```
plot_results(fit_hier_sgm_alpha,all_mus2_5,all_mus97_5,all_people)
```

In [118]:
```
new_predictions = fit_hier_sgm_alpha.extract()['y_pred_18']
new_predictions = new_predictions.T
def set_size(w,h, ax=None):
        """ w, h: width, height in inches """
        if not ax: ax=plt.gca()
        l = ax.figure.subplotpars.left
        r = ax.figure.subplotpars.right
        t = ax.figure.subplotpars.top
        b = ax.figure.subplotpars.bottom
        figw = float(w)/(r-l)
        figh = float(h)/(t-b)
        ax.figure.set_size_inches(figw, figh)

import matplotlib.gridspec as gridspec
import matplotlib.pyplot as plt
from matplotlib import figure
gs = gridspec.GridSpec(4,6)
plot_titles = ["Male 0-17","Male 18-34","Male 35-54","Male 55+",
                      "Female 0-17","Female 18-34","Female 35-54","Female 55+"]
plot_indexes = np.array([[0,3],[3,6],[0,3],[3,6],[0,3],[3,6],[0,3],[3,6]])

for i in range(4):
    plt.tight_layout(pad=0.2, w_pad=0.5, h_pad=1.0)

    ax = plt.subplot(gs[i,0:3])
    set_size(10,10)
    ax.set_ylabel("Accepted number")
    ax.set_title(plot_titles[i])
    ax.set_ylim(0,300)
    ax.hist(new_predictions[i],bins=50,ec='white')
    plt.tight_layout(pad=0.2, w_pad=0.5, h_pad=1.0)

    ax = plt.subplot(gs[i,3:6])
    set_size(10,10)
    ax.set_ylabel("Accepted number")
    ax.set_title(plot_titles[i+4])
    ax.set_ylim(0,300)
```

```
        ax.hist(new_predictions[i+4],bins=50,ec='white')
    plt.show()
```

8.8 Hierarchical model for α and σ and normal prior on mu0_alpha

In [119]:
```
population_model_hierarchical ="""
data{
    int<lower=0> N;          // number of data points
    int<lower=0> G;          // number of groups
    vector[N] years;         // years 2010...2017
    vector<lower=0>[N] y;    // the actual number of accepted citizens
    int grps_ind[N];         // group indicator vector (1,2,...,8,1,2...,8...)
    real xpred;              // year 2018

    real prior_mu;
    real<lower=0>prior_sigma;
}
parameters{
    vector[G] alpha;            // regression paramters for our
    vector[G] beta;             // data points means (separate for each age category)
    vector<lower=0>[G] sigma;   // common sigma for each group
    real mu0_alpha;
    real<lower=0>sigma0_alpha;

    real p1;
    real p2;
}
transformed parameters{
    vector[N] mu;
    for (i in 1:N)
        mu[i] =  alpha[grps_ind[i]]*years[i] + beta[grps_ind[i]];
                                            // transformed mu as a
                                            // linear function of years
}
model{
    mu0_alpha ~ normal(prior_mu,prior_sigma);
    sigma ~ cauchy(p1, p2);
                        // hierarchy on sigma
    alpha ~ normal(mu0_alpha,sigma0_alpha);
                        // hierarchy on alpha
    for (i in 1:N)
        y[i] ~ normal(mu[i],sigma[grps_ind[i]]);
                    // normal distribution around
                    // each point;
}
generated quantities {
    vector[G] y_pred_18;
    vector[N] log_lik;
```

```
    real x_pred;
    for (i in 1:G)
        y_pred_18[i] = normal_rng(alpha[grps_ind[i]]*xpred +
                                  beta[grps_ind[i]],sigma[grps_ind[i]]);

    for (i in 1:N)
        log_lik[i] = normal_lpdf(y[i] | mu[i], sigma[grps_ind[i]]);

}

"""
```

In [120]:
```
hierarchical_both_parameters = \
    pystan.StanModel(model_code=population_model_hierarchical)

years = np.array(list(range(2010,2018,1))).reshape((8,1))
years = np.tile(years,8).flatten()

hierarchical_data ={
    'N': all_people.size,
    'G':8,
    'years':years,
    'y': all_people.flatten(),
    'grps_ind': list(range(1,9,1))*8,
    'xpred': 2018,
    'prior_sigma': 400,
    'prior_mu': 21.46875
}
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_f6cca5b656ce3cc9617de1d46cf1b8fe NOW.
```

In [121]:
```
fit_hier_Nprior = hierarchical_both_parameters.sampling(data=hierarchical_data,
                  iter=1000,chains=4,control={"adapt_delta":0.95,"max_treedepth":18})
```

```
D:\Anaconda3\lib\site-packages\pystan\misc.py:399: FutureWarning: Conversion of the second argum
  elif np.issubdtype(np.asarray(v).dtype, float):
```

In [218]:
```
summary = fit_hier_Nprior.summary(pars=['alpha','beta','sigma','mu0_alpha','sigma0_alp

df = pd.DataFrame(
    summary['summary'],
    index = summary['summary_rownames'],
    columns = summary['summary_colnames']).round(2)

df.loc['alpha[0]':'alpha[7]','n_eff':'Rhat'].T
```

Out[218]:
```
       alpha[0]  alpha[1]  alpha[2]  alpha[3]  alpha[4]  alpha[5]  alpha[6]  \
n_eff    1445.0    1287.0    1516.0     720.0    1330.0    1118.0    1126.0
```

```
Rhat         1.0        1.0        1.0        1.0        1.0        1.0        1.0

       alpha[7]
n_eff    458.00
Rhat       1.01
```

```
In [219]: df.loc['beta[0]':'beta[7]','n_eff':'Rhat'].T
```

```
Out[219]:        beta[0]  beta[1]  beta[2]  beta[3]  beta[4]  beta[5]  beta[6]  beta[7]
          n_eff   1445.0   1287.0   1516.0    720.0   1330.0   1118.0   1126.0   458.00
          Rhat       1.0      1.0      1.0      1.0      1.0      1.0      1.0     1.01
```

```
In [220]: df.loc['sigma[0]':'sigma[7]','n_eff':'Rhat'].T
```

```
Out[220]:        sigma[0]  sigma[1]  sigma[2]  sigma[3]  sigma[4]  sigma[5]  sigma[6]  \
          n_eff    1196.0    1316.0    1657.0    465.00    1244.0    1143.0    1077.0
          Rhat        1.0       1.0       1.0      1.01       1.0       1.0       1.0

                 sigma[7]
          n_eff    465.00
          Rhat       1.01
```

```
In [221]: df.loc[['mu0_alpha','sigma0_alpha','p1','p2'],'n_eff':'Rhat'].T
```

```
Out[221]:        mu0_alpha  sigma0_alpha      p1      p2
          n_eff     1341.0         918.0  1497.0  623.00
          Rhat         1.0           1.0     1.0    1.01
```

```
In [236]: results = psis.psisloo(fit_hier_Nprior.extract()['log_lik'])
          log_lik = fit_hier_Nprior.extract()['log_lik']
          means = np.mean(np.exp(log_lik),axis=0)
          sums = np.sum(np.log(means))
          print(results[0])
          p_eff = sums  - results[0]
          k_vals=psis.psisloo(fit_hier_Nprior.extract()['log_lik'])[2]
          plt.hist(k_vals,bins=15,ec='white')
          plt.title("Histogram of values of K")
          plt.xlabel("K Values")
          plt.show()
```

```
-427.1931176857057
```

```
In [128]: print(p_eff)
```

```
17.460048248844203
```

```
In [129]: summary = fit_hier_Nprior.summary(pars=['mu'])


          df = pd.DataFrame(
              summary['summary'],
              index = summary['summary_rownames'],
              columns = summary['summary_colnames'])

          all_mus2_5 = []
          all_mus97_5 = []
          for i in range(0,64):
              current_mu = df.loc['mu[' + str(i)+ "]"]
              all_mus2_5.append(current_mu[3])
              all_mus97_5.append(current_mu[7])


          all_mus2_5 = np.array(all_mus2_5)
          all_mus97_5 = np.array(all_mus97_5)
          all_mus2_5 = np.reshape(all_mus2_5,(8,8))
          all_mus97_5 = np.reshape(all_mus97_5,(8,8))

          plot_results(fit_hier_Nprior,all_mus2_5,all_mus97_5,all_people)

In [130]: new_predictions = fit_hier_Nprior.extract()['y_pred_18']
          new_predictions = new_predictions.T
          def set_size(w,h, ax=None):
                  """ w, h: width, height in inches """
                  if not ax: ax=plt.gca()
                  l = ax.figure.subplotpars.left
                  r = ax.figure.subplotpars.right
                  t = ax.figure.subplotpars.top
                  b = ax.figure.subplotpars.bottom
                  figw = float(w)/(r-l)
                  figh = float(h)/(t-b)
                  ax.figure.set_size_inches(figw, figh)

          import matplotlib.gridspec as gridspec
          import matplotlib.pyplot as plt
          from matplotlib import figure
          gs = gridspec.GridSpec(4,6)
          plot_titles = ["Male 0-17","Male 18-34","Male 35-54","Male 55+",
                                "Female 0-17","Female 18-34","Female 35-54","Female 55+"]
          plot_indexes = np.array([[0,3],[3,6],[0,3],[3,6],[0,3],[3,6],[0,3],[3,6]])

          for i in range(4):
              plt.tight_layout(pad=0.2, w_pad=0.5, h_pad=1.0)

              ax = plt.subplot(gs[i,0:3])
```

```
        set_size(10,10)
        ax.set_ylabel("Accepted number")
        ax.set_title(plot_titles[i])
        ax.set_ylim(0,300)
        ax.hist(new_predictions[i],bins=50,ec='white')
        plt.tight_layout(pad=0.2, w_pad=0.5, h_pad=1.0)

        ax = plt.subplot(gs[i,3:6])
        set_size(10,10)
        ax.set_ylabel("Accepted number")
        ax.set_title(plot_titles[i+4])
        ax.set_ylim(0,300)
        ax.hist(new_predictions[i+4],bins=50,ec='white')
    plt.show()
```

8.9 Hierarchical model for α and σ and normal, more informative prior on mu0_alpha

```
In [131]: population_model_hierarchical ="""
    data{
        int<lower=0> N;         // number of data points
        int<lower=0> G;         // number of groups
        vector[N] years;        // years 2010...2017
        vector<lower=0>[N] y;   // the actual number of accepted citizens
        int grps_ind[N];        // group indicator vector (1,2,...,8,1,2...,8...)
        real xpred;             // year 2018

        real prior_mu;
        real<lower=0>prior_sigma;
    }
    parameters{
        vector[G] alpha;            // regression paramters for our
        vector[G] beta;             // data points means (separate for each age category)
        vector<lower=0>[G] sigma;   // common sigma for each group
        real mu0_alpha;
        real<lower=0>sigma0_alpha;

        real p1;
        real p2;
    }
    transformed parameters{
        vector[N] mu;
        for (i in 1:N)
            mu[i] =  alpha[grps_ind[i]]*years[i] + beta[grps_ind[i]];
                                                    // transformed mu as a
                                                    // linear function of years
    }
    model{
        mu0_alpha ~ normal(prior_mu,prior_sigma);
```

```
    sigma ~ cauchy(p1, p2);
                        // hierarchy on sigma
    alpha ~ normal(mu0_alpha,sigma0_alpha);
                        // hierarchy on alpha
    for (i in 1:N)
        y[i] ~ normal(mu[i],sigma[grps_ind[i]]);
                    // normal distribution around
                    // each point;
}
generated quantities {
    vector[G] y_pred_18;
    vector[N] log_lik;
    real x_pred;
    for (i in 1:G)
        y_pred_18[i] = normal_rng(alpha[grps_ind[i]]*xpred +
                        beta[grps_ind[i]],sigma[grps_ind[i]]);

    for (i in 1:N)
        log_lik[i] = normal_lpdf(y[i] | mu[i], sigma[grps_ind[i]]);

}

"""
```

In [132]:
```
hierarchical_both_parameters = \
    pystan.StanModel(model_code=population_model_hierarchical)

years = np.array(list(range(2010,2018,1))).reshape((8,1))
years = np.tile(years,8).flatten()

hierarchical_data ={
    'N': all_people.size,
    'G':8,
    'years':years,
    'y': all_people.flatten(),
    'grps_ind': list(range(1,9,1))*8,
    'xpred': 2018,
    'prior_sigma': 168.07,
    'prior_mu': 21.46875
}
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_f6cca5b656ce3cc9617de1d46cf1b8fe NOW.
```

In [133]:
```
fit_hier_Nprior_inf = hierarchical_both_parameters.sampling(data=hierarchical_data,
                        iter=1000,chains=4,control={"adapt_delta":0.95,"max_treedepth":18})
```

In [226]:
```
summary = fit_hier_Nprior_inf.summary(pars=['alpha','beta','sigma','mu0_alpha','sigma0
```

```
df = pd.DataFrame(
    summary['summary'],
    index = summary['summary_rownames'],
    columns = summary['summary_colnames']).round(2)

df.loc['alpha[0]':'alpha[7]','n_eff':'Rhat'].T
```

Out[226]:

```
       alpha[0]  alpha[1]  alpha[2]  alpha[3]  alpha[4]  alpha[5]  alpha[6]  \
n_eff    1180.0    1243.0    1512.0    1039.0    1429.0    1338.0    1177.0
Rhat        1.0       1.0       1.0       1.0       1.0       1.0       1.0

       alpha[7]
n_eff    320.00
Rhat       1.01
```

In [135]:

```
df.loc['beta[0]':'beta[8]','n_eff':'Rhat'].T
```

Out[135]:

```
          beta[0]      beta[1]      beta[2]      beta[3]      beta[4]  \
n_eff  1180.00000  1243.000000  1512.000000  1039.000000  1429.000000
Rhat      1.00093     0.999139     0.999697     1.002416     1.002063

           beta[5]      beta[6]     beta[7]
n_eff  1338.000000  1177.000000  320.000000
Rhat      0.998985     1.002683    1.008985
```

In [136]:

```
df.loc['sigma[0]':'sigma[8]','n_eff':'Rhat'].T
```

Out[136]:

```
          sigma[0]     sigma[1]     sigma[2]    sigma[3]     sigma[4]  \
n_eff  1310.000000  1179.000000  1139.000000  727.000000  1044.000000
Rhat      0.999295     1.000146     1.000089    1.004422     1.004493

          sigma[5]     sigma[6]    sigma[7]
n_eff  1532.000000  1233.000000  292.000000
Rhat      0.999109     1.001967    1.020195
```

In [227]:

```
df.loc[['mu0_alpha','sigma0_alpha','p1','p2'],'n_eff':'Rhat'].T
```

Out[227]:

```
       mu0_alpha  sigma0_alpha      p1      p2
n_eff     2000.0        1097.0  1386.0  629.00
Rhat         1.0           1.0     1.0    1.01
```

In [237]:

```
results = psis.psisloo(fit_hier_Nprior_inf.extract()['log_lik'])
log_lik = fit_hier_Nprior_inf.extract()['log_lik']
means = np.mean(np.exp(log_lik),axis=0)
sums = np.sum(np.log(means))
p_eff = sums  - results[0]
print(results[0])
k_vals=psis.psisloo(fit_hier_Nprior_inf.extract()['log_lik'])[2]
plt.hist(k_vals,bins=15,ec='white')
```

```
    plt.title("Histogram of values of K")
    plt.xlabel("K Values")
    plt.show()
```

```
-426.70286437238184
```

```
In [139]: print(p_eff)
```

```
16.932812959157502
```

```
In [140]: summary = fit_hier_Nprior_inf.summary(pars=['mu'])


          df = pd.DataFrame(
              summary['summary'],
              index = summary['summary_rownames'],
              columns = summary['summary_colnames'])

          all_mus2_5 = []
          all_mus97_5 = []
          for i in range(0,64):
              current_mu = df.loc['mu[' + str(i)+ "]"]
              all_mus2_5.append(current_mu[3])
              all_mus97_5.append(current_mu[7])



          all_mus2_5 = np.array(all_mus2_5)
          all_mus97_5 = np.array(all_mus97_5)
          all_mus2_5 = np.reshape(all_mus2_5,(8,8))
          all_mus97_5 = np.reshape(all_mus97_5,(8,8))

          plot_results(fit_hier_Nprior_inf,all_mus2_5,all_mus97_5,all_people)

In [141]: new_predictions = fit_hier_Nprior_inf.extract()['y_pred_18']
          new_predictions = new_predictions.T
          def set_size(w,h, ax=None):
                  """ w, h: width, height in inches """
                  if not ax: ax=plt.gca()
                  l = ax.figure.subplotpars.left
                  r = ax.figure.subplotpars.right
                  t = ax.figure.subplotpars.top
                  b = ax.figure.subplotpars.bottom
                  figw = float(w)/(r-l)
                  figh = float(h)/(t-b)
                  ax.figure.set_size_inches(figw, figh)

          import matplotlib.gridspec as gridspec
```

```
import matplotlib.pyplot as plt
from matplotlib import figure
gs = gridspec.GridSpec(4,6)
plot_titles = ["Male 0-17","Male 18-34","Male 35-54","Male 55+",
                      "Female 0-17","Female 18-34","Female 35-54","Female 55+"]
plot_indexes = np.array([[0,3],[3,6],[0,3],[3,6],[0,3],[3,6],[0,3],[3,6]])

for i in range(4):
    plt.tight_layout(pad=0.2, w_pad=0.5, h_pad=1.0)

    ax = plt.subplot(gs[i,0:3])
    set_size(10,10)
    ax.set_ylabel("Accepted number")
    ax.set_title(plot_titles[i])
    ax.set_ylim(0,300)
    ax.hist(new_predictions[i],bins=50,ec='white')
    plt.tight_layout(pad=0.2, w_pad=0.5, h_pad=1.0)

    ax = plt.subplot(gs[i,3:6])
    set_size(10,10)
    ax.set_ylabel("Accepted number")
    ax.set_title(plot_titles[i+4])
    ax.set_ylim(0,300)
    ax.hist(new_predictions[i+4],bins=50,ec='white')
plt.show()
```

8.10 Hierarchical model for α and σ and normal, more informative prior on mu0_alpha, with an intuition that the slope becomes steeper

In [143]:
```
population_model_hierarchical ="""
data{
    int<lower=0> N;          // number of data points
    int<lower=0> G;          // number of groups
    vector[N] years;         // years 2010...2017
    vector<lower=0>[N] y;    // the actual number of accepted citizens
    int grps_ind[N];         // group indicator vector (1,2,...,8,1,2...,8...)
    real xpred;              // year 2018

    real prior_mu;
    real<lower=0>prior_sigma;
}
parameters{
    vector[G] alpha;            // regression paramters for our
    vector[G] beta;             // data points means (separate for each age category)
    vector<lower=0>[G] sigma;   // common sigma for each group
    real mu0_alpha;
    real<lower=0>sigma0_alpha;
```

```
    real p1;
    real p2;
}
transformed parameters{
    vector[N] mu;
    for (i in 1:N)
        mu[i] =  alpha[grps_ind[i]]*years[i] + beta[grps_ind[i]];
                                            // transformed mu as a
                                            // linear function of years
}
model{
    mu0_alpha ~ normal(prior_mu,prior_sigma);
    sigma ~ cauchy(p1, p2);
                        // hierarchy on sigma
    alpha ~ normal(mu0_alpha,sigma0_alpha);
                        // hierarchy on alpha
    for (i in 1:N)
        y[i] ~ normal(mu[i],sigma[grps_ind[i]]);
                    // normal distribution around
                    // each point;
}
generated quantities {
    vector[G] y_pred_18;
    vector[N] log_lik;
    real x_pred;
    for (i in 1:G)
        y_pred_18[i] = normal_rng(alpha[grps_ind[i]]*xpred +
                        beta[grps_ind[i]],sigma[grps_ind[i]]);

    for (i in 1:N)
        log_lik[i] = normal_lpdf(y[i] | mu[i], sigma[grps_ind[i]]);

}

"""
```

In [144]:
```
hierarchical_both_parameters = \
    pystan.StanModel(model_code=population_model_hierarchical)

years = np.array(list(range(2010,2018,1))).reshape((8,1))
years = np.tile(years,8).flatten()

hierarchical_data ={
    'N': all_people.size,
    'G':8,
    'years':years,
    'y': all_people.flatten(),
    'grps_ind': list(range(1,9,1))*8,
```

```
        'xpred': 2018,
        'prior_sigma': 300,
        'prior_mu': 200
    }
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_f6cca5b656ce3cc9617de1d46cf1b8fe NOW.
```

```
In [145]: fit_hier_Nprior_inf_slp = hierarchical_both_parameters.sampling(data=hierarchical_data
                              iter=1000,chains=4,control={"adapt_delta":0.95,"max_treedepth":18})
```

```
In [146]: summary = fit_hier_Nprior_inf_slp.summary(pars=['alpha','beta','sigma','mu0_alpha','si

          df = pd.DataFrame(
              summary['summary'],
              index = summary['summary_rownames'],
              columns = summary['summary_colnames'])

          df.loc['alpha[0]':'alpha[8]','n_eff':'Rhat'].T
```

```
Out[146]:          alpha[0]     alpha[1]     alpha[2]    alpha[3]     alpha[4]  \
          n_eff  1434.00000  1587.000000  1488.000000  544.000000  1519.000000
          Rhat      0.99922     0.998859     0.999358    1.002304     1.001778

                    alpha[5]     alpha[6]    alpha[7]
          n_eff  1209.000000  1477.000000  299.000000
          Rhat      0.999818     1.003465    1.014105
```

```
In [147]: df.loc['beta[0]':'beta[8]','n_eff':'Rhat'].T
```

```
Out[147]:             beta[0]      beta[1]      beta[2]     beta[3]     beta[4]  \
          n_eff  1434.000000  1587.000000  1488.000000  544.000000  1519.00000
          Rhat      0.999222     0.998861     0.999357    1.002297     1.00178

                    beta[5]      beta[6]     beta[7]
          n_eff  1209.00000  1477.000000  299.000000
          Rhat      0.99982     1.003455    1.014105
```

```
In [148]: df.loc['sigma[0]':'sigma[8]','n_eff':'Rhat'].T
```

```
Out[148]:           sigma[0]     sigma[1]     sigma[2]    sigma[3]     sigma[4]  \
          n_eff  1129.000000  2000.000000  1473.000000  381.000000  1183.000000
          Rhat      0.999977     1.000255     1.002602    1.018665     0.999861

                    sigma[5]     sigma[6]    sigma[7]
          n_eff  1666.000000  1204.000000  331.000000
          Rhat      1.003505     1.002995    1.022896
```

```
In [228]: df.loc[['mu0_alpha','sigma0_alpha','p1','p2'],'n_eff':'Rhat'].T
```

Out[228]:
```
      mu0_alpha  sigma0_alpha      p1      p2
n_eff    2000.0        1097.0  1386.0  629.00
Rhat        1.0           1.0     1.0    1.01
```

In [238]:
```
results = psis.psisloo(fit_hier_Nprior_inf_slp.extract()['log_lik'])
log_lik = fit_hier_Nprior_inf_slp.extract()['log_lik']
means = np.mean(np.exp(log_lik),axis=0)
sums = np.sum(np.log(means))
p_eff = sums  - results[0]
print(results[0])
k_vals=psis.psisloo(fit_hier_Nprior_inf_slp.extract()['log_lik'])[2]
plt.hist(k_vals,bins=15,ec='white')
plt.title("Histogram of values of K")
plt.xlabel("K Values")
plt.show()
```

```
-427.24115519392996
```

In [150]:
```
print(p_eff)
```

```
17.39036335611962
```

In [152]:
```
summary = fit_hier_Nprior_inf_slp.summary(pars=['mu'])


df = pd.DataFrame(
    summary['summary'],
    index = summary['summary_rownames'],
    columns = summary['summary_colnames'])

all_mus2_5 = []
all_mus97_5 = []
for i in range(0,64):
    current_mu = df.loc['mu[' + str(i)+ "]"]
    all_mus2_5.append(current_mu[3])
    all_mus97_5.append(current_mu[7])


all_mus2_5 = np.array(all_mus2_5)
all_mus97_5 = np.array(all_mus97_5)
all_mus2_5 = np.reshape(all_mus2_5,(8,8))
all_mus97_5 = np.reshape(all_mus97_5,(8,8))

plot_results(fit_hier_Nprior_inf_slp,all_mus2_5,all_mus97_5,all_people)
```

In [153]:
```
new_predictions = fit_hier_Nprior_inf_slp.extract()['y_pred_18']
new_predictions = new_predictions.T
```

```
def set_size(w,h, ax=None):
        """ w, h: width, height in inches """
        if not ax: ax=plt.gca()
        l = ax.figure.subplotpars.left
        r = ax.figure.subplotpars.right
        t = ax.figure.subplotpars.top
        b = ax.figure.subplotpars.bottom
        figw = float(w)/(r-l)
        figh = float(h)/(t-b)
        ax.figure.set_size_inches(figw, figh)

import matplotlib.gridspec as gridspec
import matplotlib.pyplot as plt
from matplotlib import figure
gs = gridspec.GridSpec(4,6)
plot_titles = ["Male 0-17","Male 18-34","Male 35-54","Male 55+",
                      "Female 0-17","Female 18-34","Female 35-54","Female 55+"]
plot_indexes = np.array([[0,3],[3,6],[0,3],[3,6],[0,3],[3,6],[0,3],[3,6]])

for i in range(4):
    plt.tight_layout(pad=0.2, w_pad=0.5, h_pad=1.0)

    ax = plt.subplot(gs[i,0:3])
    set_size(10,10)
    ax.set_ylabel("Accepted number")
    ax.set_title(plot_titles[i])
    ax.set_ylim(0,300)
    ax.hist(new_predictions[i],bins=50,ec='white')
    plt.tight_layout(pad=0.2, w_pad=0.5, h_pad=1.0)

    ax = plt.subplot(gs[i,3:6])
    set_size(10,10)
    ax.set_ylabel("Accepted number")
    ax.set_title(plot_titles[i+4])
    ax.set_ylim(0,300)
    ax.hist(new_predictions[i+4],bins=50,ec='white')
plt.show()
```

9 Appendix

```
In [176]: def plot_results_pooled(fit,all_mus2_5,all_mus97_5,all_people):
              import numpy as np
              alpha = fit.extract('alpha')
              alpha = alpha['alpha']
              alpha = np.mean(alpha,axis=0)
              beta = fit.extract('beta')
              beta = beta['beta']
```

```
            beta = np.mean(beta,axis=0)
            new_predictions = fit.extract()['y_pred_18']
            new_predictions = np.mean(new_predictions,axis=0)
            import matplotlib.pyplot as plt
            years_repeat = np.array(list(range(2010,2018,1))).reshape((8,1))
            years_repeat = np.tile(years_repeat,8).flatten()
            years = np.array(list(range(2010,2018,1)))

            flatten_all_ppl = all_people.flatten()
            plt.scatter(years_repeat,flatten_all_ppl)
            pred_vals = []
            for year in years:
                pred_vals.append(alpha*year+beta)
            plt.plot(years,pred_vals)
            plt.plot(years,all_mus2_5,linestyle='--')
            plt.plot(years,all_mus97_5,linestyle='--')
```

```
In [177]: def plot_results(fit,all_mus2_5,all_mus97_5,all_people):
              import numpy as np
              alpha = fit.extract('alpha')
              alpha = alpha['alpha']
              a = np.mean(alpha,axis=0)
              b = fit.extract('beta')
              b = b['beta']
              b = np.mean(b,axis=0)
              new_predictions = fit.extract()['y_pred_18']
              new_predictions = np.mean(new_predictions,axis=0)
              def set_size(w,h, ax=None):
                  """ w, h: width, height in inches """
                  if not ax: ax=plt.gca()
                  l = ax.figure.subplotpars.left
                  r = ax.figure.subplotpars.right
                  t = ax.figure.subplotpars.top
                  b = ax.figure.subplotpars.bottom
                  figw = float(w)/(r-l)
                  figh = float(h)/(t-b)
                  ax.figure.set_size_inches(figw, figh)

              import matplotlib.gridspec as gridspec
              import matplotlib.pyplot as plt
              from matplotlib import figure
              gs = gridspec.GridSpec(4,6)
              plot_titles = ["Male 0-17","Male 18-34","Male 35-54","Male 55+",
                             "Female 0-17","Female 18-34","Female 35-54","Female 55+"]
              plot_indexes = np.array([[0,3],[3,6],[0,3],[3,6],[0,3],[3,6],[0,3],[3,6]])
              years = list(range(2010,2018,1))
              years = np.array(years)
              plt.show()
```

```
for i in range(4):

        # PLOT MALE
    if (i < 3):
        male = all_people[:,i]
        female = all_people[:,i+4]
        ax = plt.subplot(gs[i,0:3])
        set_size(10,10)
        plt.tight_layout(pad=0.2, w_pad=0.5, h_pad=1.0)
        ax.set_xlabel("Years")
        ax.set_ylabel("Accepted number")
        ax.set_title(plot_titles[i])
        ax.set_ylim(0,2700)
        ax.scatter(years,male)
        ax.plot(years,all_mus2_5[:,i],linestyle='--')

        ax.plot(years,b[i]+ a[i]*years) # ???
        ax.plot(years,all_mus97_5[:,i],linestyle='--')
        ax.scatter(2018,new_predictions[i],marker="x")
        # PLOT FEMALE
        ax = plt.subplot(gs[i,3:6])
        ax.set_xlabel("Years")
        ax.set_ylabel("Accepted number")
        ax.set_title(plot_titles[i+4])
        ax.set_ylim(0,2700)
        ax.scatter(years,female)
        ax.plot(years,all_mus2_5[:,i+4],linestyle='--')
        ax.plot(years,b[i+4]+a[i+4]*years)
        ax.scatter(2018,new_predictions[i+4],marker="x")
        ax.plot(years,all_mus97_5[:,i+4],linestyle='--')
    else:
        male = all_people[:,i]
        female = all_people[:,i+4]
        ax = plt.subplot(gs[i,0:3])
        set_size(10,10)
        plt.tight_layout(pad=0.2, w_pad=0.5, h_pad=1.0)
        ax.set_xlabel("Years")
        ax.set_ylabel("Accepted number")
        ax.set_title(plot_titles[i])
        ax.set_ylim(0,500)
        ax.scatter(years,male)
        ax.plot(years,all_mus2_5[:,i],linestyle='--')
        ax.plot(years,b[i]+a[i]*years)
        ax.plot(years,all_mus97_5[:,i],linestyle='--')
        ax.scatter(2018,new_predictions[i],marker="x")
        # PLOT FEMALE
        ax = plt.subplot(gs[i,3:6])
        ax.set_xlabel("Years")
```

```
        ax.set_ylabel("Accepted number")
        ax.set_title(plot_titles[i+4])
        ax.set_ylim(0,500)
        ax.scatter(years,female)
        ax.plot(years,all_mus2_5[:,i+4],linestyle='--')
        ax.plot(years,b[i+4]+a[i+4]*years)
        ax.scatter(2018,new_predictions[i+4],marker="x")
        ax.plot(years,all_mus97_5[:,i+4],linestyle='--')
plt.show()
```

10 References

1. Bayesian statistics using STAN
2. Common variance (ANOVA) model
3. Cauchy Distribution
4. Immigration statistics
5. Prior choice analysis
6. Bayesian Linear Regression Models
7. Bayesian Data Analysis, Third Edition, by Andrew Gelman, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari, Donald B. Rubin