MAS 4106 Final Project

Authors: Benton Stacy (bmstacy7127@eagle.fgcu.edu) and Katarya Johnson-Williams (kajohnsonwilliam3168@eagle.fgcu.edu)

Abstract: The reason for writing this report is to explore the possible ways to predict years of education using census data. The processes and methods developed in this report could be applied to future census data to create an effective method of predicting significant census attributes. The problem this report seeks to solve is identifying patterns in the data provided by the census that can be used to predict years of education an individual has completed. A secondary problem the methodology in this report can be used to solve is to fill in missing data points in the census data. This report creates ten models that analyze the 15 different attributes included with the data in addition to adding two additional attributes (gross domestic product and Human Development Index for native countries). The models use a least squares solution which is validated by cross-validation. The results of this report did not find high variance in the error of each model but did make interesting discoveries regarding the relationship between the attributes indicating education and the native country of an individual. The overall conclusion made is that the United States population is significantly diverse and therefore it is difficult to accurately predict values such as years of education. The results of this report leave many avenues for future work including applying the models created to modern census data or adding additional census attributes (i.e. number of children or parental education levels) to see how these attributes affect the model.

Initialize the Dataset

In [1]:
```julia
# Load packages
using CSV, DataFrames, LinearAlgebra, Missings # For dataset loading and processing, running least-squares calculations.
using Plots, Measures, StatsPlots # For plotting results. Measures used for better plot margins.
using Random # For generating random permutations.
using LaTeXStrings # For generating beautiful TeX-like equations in plots.

# Set up a few variables for the filepath of this file.
# NOTE: You may need to change this directory based on where your dataset is stored.
    # By default, it assumes the dataset is stored in (root)\Data Sets\Adult
root = dirname(@__FILE__)
filename = "adult_full.csv";
filepath = joinpath(root, "Data Sets", "Adult", filename);

# Read the dataset from the filepath.
adultDataSet = CSV.read(filepath,DataFrame;header=true, missingstring="?");
# There aren't too many missing values, so drop them.
adultDataSet = dropmissing(adultDataSet, disallowmissing=true)

# Number of entries in the dataset
m = length(adultDataSet[:,1])
```

Out[1]:
```
45222
```

Construction of the Columns

The original dataset, after removing missing entries, has 45222 entries and 15 attributes, namely:

1. Age – age of the individual. This data is numerical, in whole numbers of years.
2. Work Class – sector the individual works in (government, private, self-employed, not working). The mode of this attribute was private, with nearly 75% of the entries.
3. Final Weight – a (continuous) number estimating the number of people in the United States population which matches the individual's demographics. This value considers the potential for sampling bias in the census and assigns a "weight" to the person based on their demographics. Final weight may be calculated at the federal, state, or local level and thus values in this category may not be consistent.
4. Education – label describing the highest level of education completed.
5. Education Number – arbitrary number assigned according to education label.
6. Marital Status – current marital status of the individual (single, married, previously married).
7. Occupation – a variety of labels of the general job title of the individual.
8. Relationship – a label of the relationship the individual has (wife, husband, unmarried, child, not in family).
9. Race – the race of the individual (white, Asian/Pacific Islander, Native American ("American Indian/Eskimo" in the data), black, other).
10. Sex – the sex of the individual (male, female).
11. Capital Gain – a continuous category which measures an individual's capital gain. Note that this is different from income and might include income earned via stocks and investments.
12. Capital Loss – a continuous category which measures an individual's capital loss.
13. Hours Worked per Week – a numerical category which details the number of hours per week an individual works.
14. Native Country – a label describing an individual's native country. This attribute is highly skewed, with the mode of United States comprising of over 91% of the data.
15. Income Level – a Boolean (true/false) attribute which describes if the individual made a yearly income of greater than $50,000.

To process the data for purposes of creating and testing the models, the columns were modified as follows:

- Removed attributes:
  - Final weight – found to be inconsistently calculated across censuses, states, and local jurisdictions. In addition, information on final weight in the dataset documentation lacked sufficient detail.
  - Capital gain and capital loss – the original dataset documentation lacked sufficient detail on these columns. Also, theese attributes are *highly* skewed towards zero, thus providing little prediction power.

    - Education number – the original values seemed arbitrary. These values were replaced with a custom scale.
    - Relationship – little detail found in the original documentation. Similar in nature to the marital status attribute so that was used instead.
- Modified attributes:
    - Work class – separated into three columns based on the economic sector the individual is employed in: private sector, self-employed, or public (government). A fourth implicit column is used for those not working.
    - Education – the values were replaced with a scale based on the typical number of years an individual of that category would be expected to have completed. For example, a value of `10th` likely indicates around 11 years of schooling.
    - Marital status – labels were separated into two columns to represent single and married individuals. An implicit third column is used for those who have been previously married.
    - Occupation – labels were assigned one of five general occupational categories: engineering, business, technical, non-degree, and governmental jobs. The first four categories were assigned a column and the last one (government) was implicit.
    - Race – labels were assigned into four separate columns: white, Asian / Pacific-Islander, Native American (represented as `Amer-Indian-Eskimo`), and black. A fifth implicit column is designated fore those with the race label `other`.
    - Sex – transformed from a boolean variable into a numnerical one. Male is represented as "0" and female is represented as "1".
    - Native country – labels were assigned one of four categories based on the continent the country is located in: North America, South America, Asia, and Europe. The "Europe" column is implicit.
- Added attributes:
    - Gross domestic product (GDP) – a continuous value which represents the market value of the country's total goods and services produced over the year. The column consists of each applicable native country's 1994 GDP in $ millions USD, not accounting for inflation. GDP can be used as a secondary economic indicator for an individual by proxy of their native country.
    - Human development index (HDI) – an aggregate value on a continuous scale from 0 to 1, where higher values represent a higher level of human development. The formula for HDI calculations involve expected education levels, the country's GDP, and life expectancy. 1990 HDI data was extracted for the purposes of this program.

The revised dataset has 12 attributes. Attributes of more than one column have their columns listed. Implicit columns are *italicized*.

1. Age
2. Work Class
    - Private
    - Self-employed
    - Government
    - *Not working*

3. Education (years)
4. Marital Status
   - Single
   - Married
   - *Previously married*
5. Occupation
   - Engineering
   - Business
   - Technical
   - Non-degree
   - *Government*
6. Race
   - White
   - Asian / Pacific-Islander
   - Native American
   - Black
   - *Other*
7. Sex (Boolean)
8. Hours worked per week
9. Native Country
   - North America
   - South America
   - Asia
   - *Europe*
10. Income (Boolean)
11. GDP of native country
12. HDI of native country

In [2]:
```
# Age
ageClass = adultDataSet[:,1];

# Work Class
```

```
    # 100: Private
    # 010: Self-employed
    # 001: Government
    # 000: Not working
workClass1 = adultDataSet[:,2].== " Private"

workClass21 = adultDataSet[:,2].== " Self-emp-not-inc"
workClass22 = adultDataSet[:,2].== " Self-emp-inc"

workClass31 = adultDataSet[:,2].== " Federal-gov"
workClass32 = adultDataSet[:,2].== " Local-gov"
workClass33 = adultDataSet[:,2].== " State-gov"

workClassMatrix = [workClass1 workClass21+workClass22 workClass31+workClass32+workClass33];

# Education
eduClass1 = (adultDataSet[:,4].== " Doctorate") .* 24
eduClass2 = (adultDataSet[:,4].== " Masters") .* 19
eduClass3 = (adultDataSet[:,4].== " Bachelors") .*17
eduClass4 = (adultDataSet[:,4].== " Some-college") .*14
eduClass5 = (adultDataSet[:,4].== " HS-grad") .*13
eduClass6 = (adultDataSet[:,4].== " 12th") .*13
eduClass7 = (adultDataSet[:,4].== " 11th") .* 12
eduClass8 = (adultDataSet[:,4].== " 10th") .* 11
eduClass9 = (adultDataSet[:,4].== " 9th") .* 10
eduClass10 = (adultDataSet[:,4].== " 7th-8th") .* 8
eduClass11 = (adultDataSet[:,4].== " 5th-6th") .* 6
eduClass12 = (adultDataSet[:,4].== " 1st-4th") .* 3
eduClass13 = (adultDataSet[:,4].== " Preschool")
eduClass14 = (adultDataSet[:,4].== " Prof-school") .* 15
eduClass15 = (adultDataSet[:,4].== " Assoc-acdm") .* 15
eduClass16 = (adultDataSet[:,4].== " Assoc-voc") .* 15

eduClass  = (eduClass1+eduClass2+eduClass3+eduClass4+eduClass5+eduClass6+eduClass7+eduClass8+eduClass9+
    eduClass10+ eduClass11+eduClass12+eduClass13+eduClass14+eduClass15+eduClass16);

# Marital Status
    # 10: Single
    # 01: Married
    # 00: Previously married

marryClass1 = adultDataSet[:,6].== " Never-married"
marryClass21 = adultDataSet[:,6].== " Married-civ-spouse"
```

```
marryClass22 = adultDataSet[:,6].== " Married-spouse-absent"

marryClass23 = adultDataSet[:,6].== " Married-AF-spouse"

marryMatrix = [marryClass1 marryClass21+marryClass22+marryClass23];

# Occupation
    # 1000: Engineering
    # 0100: Business
    # 0010: Technical
    # 0001: Non-degree
    # 0000: Government

occClass11 = adultDataSet[:,7].== " Tech-support"
occClass12 = adultDataSet[:,7].== " Machine-op-inspct"
occClass21 = adultDataSet[:,7].== " Sales"
occClass22 = adultDataSet[:,7].== " Exec-managerial"
occClass23 = adultDataSet[:,7].== " Adm-clerical"
occClass31 = adultDataSet[:,7].== " Craft-repair"
occClass32 = adultDataSet[:,7].== " Prof-specialty"
occClass41 = adultDataSet[:,7].== " Other-service"
occClass42 = adultDataSet[:,7].== " Handlers-cleaners"
occClass43 = adultDataSet[:,7].== " Farming-fishing"
occClass44 = adultDataSet[:,7].== " Transport-moving"

occMatrix = [occClass11+occClass12 occClass21+occClass22+occClass23 occClass31+occClass32 occClass41+occClass42+
    occClass43+occClass44];

# Race
raceClass1 = adultDataSet[:,9].== " White"
raceClass2 = adultDataSet[:,9].== " Asian-Pac-Islander"
raceClass3 = adultDataSet[:,9].== " Amer-Indian-Eskimo"
raceClass4 = adultDataSet[:,9].== " Black"

raceMatrix = [raceClass1 raceClass2 raceClass3 raceClass4];

# Sex
sexClass = adultDataSet[:,10].== " Female";

# Hours Per Week
hrsPerWeekClass = adultDataSet[:,13];
```

```
# Native Country
    # 100: North America
    # 010: South America
    # 001: Asia
    # 000: Europe
natClass11 = adultDataSet[:,14].== " United-States"
natClass12 = adultDataSet[:,14].== " Outlying-US(Guam-USVI-etc)"
natClass13 = adultDataSet[:,14].== " Puerto-Rico"
natClass14 = adultDataSet[:,14].== " Canada"
natClass15 = adultDataSet[:,14].== " Cuba"
natClass16 = adultDataSet[:,14].== " Honduras"
natClass17 = adultDataSet[:,14].== " Jamaica"
natClass18 = adultDataSet[:,14].== " Mexico"
natClass19 = adultDataSet[:,14].== " Dominican-Republic"
natClass110 = adultDataSet[:,14].== " Haiti"
natClass111 = adultDataSet[:,14].== " Nicaragua"
natClass112 = adultDataSet[:,14].== " El-Salvador"
natClass113 = adultDataSet[:,14].== " Trinidad&Tobago"

natClass21 = adultDataSet[:,14].== " Ecuador"
natClass22 = adultDataSet[:,14].== " Columbia"
natClass23 = adultDataSet[:,14].== " Guatemala"
natClass24 = adultDataSet[:,14].== " Peru"

natClass31 = adultDataSet[:,14].== " Cambodia"
natClass32 = adultDataSet[:,14].== " India"
natClass33 = adultDataSet[:,14].== " Japan"
natClass34 = adultDataSet[:,14].== " South"
natClass35 = adultDataSet[:,14].== " China"
natClass36 = adultDataSet[:,14].== " Iran"
natClass37 = adultDataSet[:,14].== " Philippines"
natClass38 = adultDataSet[:,14].== " Vietnam"
natClass39 = adultDataSet[:,14].== " Laos"
natClass310 = adultDataSet[:,14].== " Taiwan"
natClass311 = adultDataSet[:,14].== " Thailand"
natClass312 = adultDataSet[:,14].== " Hong"

natMatrix = [(natClass11+natClass12+natClass13+natClass14+natClass15+natClass16+natClass17+natClass18+natClass19+
        natClass110+natClass111+natClass112+natClass113) (natClass21+natClass22+natClass23+natClass24) (natClass31+
        natClass32+natClass33+natClass34+natClass35+natClass36+natClass37+natClass38+natClass39+natClass310+
        natClass311+natClass312)];

# Native Country GDP
```

```
# Native Country GDP
# Numbers are in Millions of USD
# GDP data from 1994
# Source: https://countryeconomy.com/gdp?year=1994
gdpClass1 = (adultDataSet[:,14].== " United-States") .* 7287200
gdpClass2 = (adultDataSet[:,14].== " Outlying-US(Guam-USVI-etc)") .* 7287200 # Uses US GDP
gdpClass3 = (adultDataSet[:,14].== " Puerto-Rico") .* 7287200 # Uses US GDP
gdpClass4 = (adultDataSet[:,14].== " Canada") .* 579913
gdpClass5 = (adultDataSet[:,14].== " Cuba") .* 28448
gdpClass6 = (adultDataSet[:,14].== " Honduras") .* 4642
gdpClass7 = (adultDataSet[:,14].== " Jamaica") .* 5453
gdpClass8 = (adultDataSet[:,14].== " Mexico") .* 527811
gdpClass9 = (adultDataSet[:,14].== " Dominican-Republic") .* 14645
gdpClass10 = (adultDataSet[:,14].== " Haiti") .* 3054
gdpClass11 = (adultDataSet[:,14].== " Nicaragua") .* 3861
gdpClass12 = (adultDataSet[:,14].== " El-Salvador") .* 7679
gdpClass13 = (adultDataSet[:,14].== " Trinadad&Tobago") .* 5032
gdpClass14 = (adultDataSet[:,14].== " Ecuador") .* 21147
gdpClass15 = (adultDataSet[:,14].== " Columbia") .* 97625
gdpClass16 = (adultDataSet[:,14].== " Guatemala") .* 12501
gdpClass17 = (adultDataSet[:,14].== " Peru") .* 43225
gdpClass18 = (adultDataSet[:,14].== " Cambodia") .* 2765
gdpClass19 = (adultDataSet[:,14].== " India") .* 333014
gdpClass20 = (adultDataSet[:,14].== " Japan") .* 4998797
gdpClass21 = (adultDataSet[:,14].== " South") .* 463520
gdpClass22 = (adultDataSet[:,14].== " China") .* 561686
gdpClass23 = (adultDataSet[:,14].== " Iran") .* 79818
gdpClass24 = (adultDataSet[:,14].== " Philippines") .* 73159
gdpClass25 = (adultDataSet[:,14].== " Vietnam") .* 20712
gdpClass26 = (adultDataSet[:,14].== " Laos") .* 3081
gdpClass27 = (adultDataSet[:,14].== " Taiwan") .* 256247
gdpClass28 = (adultDataSet[:,14].== " Thailand") .* 146684
gdpClass29 = (adultDataSet[:,14].== " Hong") .* 135812
gdpClass30 = (adultDataSet[:,14].== " England") .* 1244009
gdpClass31 = (adultDataSet[:,14].== " Germany") .* 2209934
gdpClass32 = (adultDataSet[:,14].== " Greece") .* 115694
gdpClass33 = (adultDataSet[:,14].== " Italy") .* 1088506
gdpClass34 = (adultDataSet[:,14].== " Poland") .* 103887
gdpClass35 = (adultDataSet[:,14].== " Portugal") .* 99692
gdpClass36 = (adultDataSet[:,14].== " Ireland") .* 55843
gdpClass37 = (adultDataSet[:,14].== " France") .* 1396653
gdpClass38 = (adultDataSet[:,14].== " Hungary") .* 43167
gdpClass39 = (adultDataSet[:,14].== " Scotland") .* 1244009 # Uses England's GDP
```

```
# Yugoslavia was split into Bosnia and Herzegovina, Croatia, Macedonia, Montenegro,
# Serbia, Slovenia, and Kosovo (not in data) in 1991 so these GDPs were all added together
gdpClass40 = (adultDataSet[:,14].== " Yugoslavia") .* 60158
gdpClass41 = (adultDataSet[:,14].== " Holand-Netherlands") .* 382550

gdpClass = (gdpClass1+gdpClass2+gdpClass3+gdpClass4+gdpClass5+gdpClass6+gdpClass7+gdpClass8+gdpClass9+gdpClass10+
    gdpClass11+gdpClass12+gdpClass13+gdpClass14+gdpClass15+gdpClass16+gdpClass17+gdpClass18+gdpClass19+gdpClass20+
    gdpClass21+gdpClass22+gdpClass23+gdpClass24+gdpClass25+gdpClass26+gdpClass27+gdpClass28+gdpClass29+gdpClass30+
    gdpClass31+gdpClass32+gdpClass33+gdpClass34+gdpClass35+gdpClass36+gdpClass37+gdpClass38+gdpClass39+gdpClass40+
    gdpClass41);


# Native Country HDI
# HDI, or Human Development Index, is a statistic of human development, broken down into
    # life expectancy;
    # education;
    # income.
# HDI data from 1990.
# Source: https://hdr.undp.org/data-center/documentation-and-downloads
hdiClass1 = (adultDataSet[:,14].== " United-States") .* 0.872
hdiClass2 = (adultDataSet[:,14].== " Outlying-US(Guam-USVI-etc)") .* 0.872 # Uses US HDI
hdiClass3 = (adultDataSet[:,14].== " Puerto-Rico") .* 0.872 # Uses US HDI
hdiClass4 = (adultDataSet[:,14].== " Canada") .* 0.860
hdiClass5 = (adultDataSet[:,14].== " Cuba") .* 0.680
hdiClass6 = (adultDataSet[:,14].== " Honduras") .* 0.516
hdiClass7 = (adultDataSet[:,14].== " Jamaica") .* 0.659
hdiClass8 = (adultDataSet[:,14].== " Mexico") .* 0.662
hdiClass9 = (adultDataSet[:,14].== " Dominican-Republic") .* 0.577
hdiClass10 = (adultDataSet[:,14].== " Haiti") .* 0.429
hdiClass11 = (adultDataSet[:,14].== " Nicaragua") .* 0.490
hdiClass12 = (adultDataSet[:,14].== " El-Salvador") .* 0.525
hdiClass13 = (adultDataSet[:,14].== " Trinadad&Tobago") .* 0.660
hdiClass14 = (adultDataSet[:,14].== " Ecuador") .* 0.651
hdiClass15 = (adultDataSet[:,14].== " Columbia") .* 0.610
hdiClass16 = (adultDataSet[:,14].== " Guatemala") .* 0.484
hdiClass17 = (adultDataSet[:,14].== " Peru") .* 0.621
hdiClass18 = (adultDataSet[:,14].== " Cambodia") .* 0.378
hdiClass19 = (adultDataSet[:,14].== " India") .* 0.434
hdiClass20 = (adultDataSet[:,14].== " Japan") .* 0.845
hdiClass21 = (adultDataSet[:,14].== " South") .* 0.737
hdiClass22 = (adultDataSet[:,14].== " China") .* 0.484
hdiClass23 = (adultDataSet[:,14].== " Iran") .* 0.601
```

Out[2]:

```
hdiClass24 = (adultDataSet[:,14].== " Philippines") .* 0.598
hdiClass25 = (adultDataSet[:,14].== " Vietnam") .* 0.482
hdiClass26 = (adultDataSet[:,14].== " Laos") .* 0.405
hdiClass27 = (adultDataSet[:,14].== " Taiwan") .* 0.484 # Uses China's HDI
hdiClass28 = (adultDataSet[:,14].== " Thailand") .* 0.576
hdiClass29 = (adultDataSet[:,14].== " Hong") .* 0.788
hdiClass30 = (adultDataSet[:,14].== " England") .* 0.804
hdiClass31 = (adultDataSet[:,14].== " Germany") .* 0.829
hdiClass32 = (adultDataSet[:,14].== " Greece") .* 0.759
hdiClass33 = (adultDataSet[:,14].== " Italy") .* 0.778
hdiClass34 = (adultDataSet[:,14].== " Poland") .* 0.716
hdiClass35 = (adultDataSet[:,14].== " Portugal") .* 0.701
hdiClass36 = (adultDataSet[:,14].== " Ireland") .* 0.737
hdiClass37 = (adultDataSet[:,14].== " France") .* 0.791
hdiClass38 = (adultDataSet[:,14].== " Hungary") .* 0.720
hdiClass39 = (adultDataSet[:,14].== " Scotland") .* 0.804 # Uses England's HDI
# Yugoslavia was split into Bosnia and Herzegovina, Croatia, Macedonia, Montenegro,
# Serbia, Slovenia, and Kosovo (not in data) in 1991 so these HDIs were (weighted per capita) averaged
hdiClass40 = (adultDataSet[:,14].== " Yugoslavia") .* 0.714
hdiClass41 = (adultDataSet[:,14].== " Holand-Netherlands") .* 0.847

hdiClass = (hdiClass1+hdiClass2+hdiClass3+hdiClass4+hdiClass5+hdiClass6+hdiClass7+hdiClass8+hdiClass9+hdiClass10+
    hdiClass11+hdiClass12+hdiClass13+hdiClass14+hdiClass15+hdiClass16+hdiClass17+hdiClass18+hdiClass19+hdiClass20+
    hdiClass21+hdiClass22+hdiClass23+hdiClass24+hdiClass25+hdiClass26+hdiClass27+hdiClass28+hdiClass29+hdiClass30+
    hdiClass31+hdiClass32+hdiClass33+hdiClass34+hdiClass35+hdiClass36+hdiClass37+hdiClass38+hdiClass39+hdiClass40+
    hdiClass41);


# Income
incomeClass = adultDataSet[:,15].== " <=50K";

# Create Matrix
```

In [3]:

```
# Create the histogram for age data and save it
ageHist = histogram(adultDataSet[:, 1], bins=15, lab="Age", xlab="Age (years)", ylab="Frequency",
    title="Histogram of Age Data  " * L"(n=%$m)", color=:blues)
savefig(ageHist, "Age_statistics.png")
```

Out[3]:

```
"C:\\Users\\bento\\OneDrive\\School Documents\\Spring 2023\\MAS 4106\\Final Project\\Age_statistics.png"
```

Work Class

In [4]:
```
# Create proprtions for the work class categories:
    # y[1]: Private
    # y[2]: Self-employed
    # y[3]: Government
    # y[4]: Not working
y = zeros(4);
y[1] = count(workClassMatrix[:, 1].==1)/m;
y[2] = count(workClassMatrix[:, 2].==1)/m;
y[3] = count(workClassMatrix[:, 3].==1)/m;
y[4] = 1 - y[1] - y[2] - y[3];

# Set up data labels for the pie chart
x = ["Private", "Self-employed", "Government", "Not working"]

# Create the pie chart for work class data and save it
workClassPie = pie(x, y, title = "Work Class  " * L"(n=%$m)")
savefig(workClassPie, "Work_class_statistics.png")
```

Out[4]:
```
"C:\\Users\\bento\\OneDrive\\School Documents\\Spring 2023\\MAS 4106\\Final Project\\Work_class_statistics.png"
```

Native Country

In [5]:
```julia
# Create proportions for the native country categories:
    # y[1]: United States
    # y[2]: The rest of North America
    # y[3]: South America
    # y[4]: Asia
    # y[5]: Europe
y = zeros(5);
y[1] = count(adultDataSet[:,14].== " United-States") / m
y[2] = count(natClass12+natClass13+natClass14+natClass15+natClass16+natClass17+natClass18+natClass19+
    natClass110+natClass111+natClass112+natClass113 .== 1) / m
y[3] = count(natClass21+natClass22+natClass23+natClass24 .== 1) / m
y[4] = count(natClass31+natClass32+natClass33+natClass34+natClass35+natClass36+natClass37+natClass38+
    natClass39+natClass310+natClass311+natClass312 .== 1) / m
y[5] = 1 - y[1] - y[2] - y[3] - y[4];

# Set up data labels for the pie chart
x = ["United States", "North America*", "South America", "Asia", "Europe"]

# Create the pie chart for native country data and save it
workClassPie = pie(x, y, title = "Native Country  " * L"(n=%$m)", color=["brown", "brown1", "chartreuse2",
        "gold", "deepskyblue"])
annotate!(0.65, -1.1, [text("* The rest of North America, excluding the United States", 8)])
savefig(workClassPie, "Native_country_statistics.png")
```

Out[5]:
```
"C:\\Users\\bento\\OneDrive\\School Documents\\Spring 2023\\MAS 4106\\Final Project\\Native_country_statistics.png"
```

Exploratory Models

Model 1: Age & Hours Worked Per Week

In [6]:

```
# Create the matrix for model 1
Amod1 = [ones(m) ageClass hrsPerWeekClass]

# Create a number storing the number of elements in each fold
fold = div(m,5);

# Create a random permutation of m-values
I = Random.randperm(m);

# Storage for cross-validation results
xhats1 = zeros(3,5);
rmsErrors1 = zeros(2,5);

# For each fold, compute the appropriate model and store its error.
anim = @animate for k = 1:5

    # Assign a random permutation into training and test data
    if (k == 1)
        Itest = I[1 : fold];
        Itrain = I[fold+1 : end];
    elseif (k == 5)
        Itest = I[4 * fold+1 : end];
        Itrain = I[1:4 * fold];
    else
        Itest = I[(k-1) * fold+1: k*fold];
        Itrain = I[[1 : (k-1) * fold ; k * fold + 1 : m]];
    end

    #display(Itest);
    #display(Itrain);

    # Compute sample sizes for training and test data
    mTest = length(Itest);
    mTrain = length(Itrain);

    # Compute the model based on training data and store it
    xhats1[:, k] = Amod1[Itrain, :] \ eduClass[Itrain];

    # Compute RMS errors for both training and test data
    rmsErrors1[1,k] = norm(Amod1[Itrain,:] * xhats1[:, k] - eduClass[Itrain]) / sqrt(mTrain);
    rmsErrors1[2,k] = norm(Amod1[Itest,:] * xhats1[:, k] - eduClass[Itest]) / sqrt(mTest);
```

```julia
    ## Plot actual (x) versus predicted (y) education
    # Plot test data
    tp1 = scatter(eduClass[Itest], Amod1[Itest, :] * xhats1[:, k], color="seagreen3", xlims=(0,25), ylims=(0,25),
        title="Test data  " * L"(n=%$mTest)", xlabel="Actual education (years)",
        ylabel="Predicted education (years)", minorgrid=true)
    plot!([0,25], [0,25], linestyle = :dash, linewidth=2, color="magenta") # y=x line

    # Plot training data
    p1 = scatter(eduClass[Itrain], Amod1[Itrain, :] * xhats1[:, k], color="seagreen3", xlims=(0,25), ylims=(0,25),
        title="Training data  " * L"(n=%$mTrain)", xlabel="Actual education (years)",
        ylabel="Predicted education (years)", minorgrid=true)
    plot!([0,25], [0,25], linestyle = :dash, linewidth=2, color="magenta") # y=x line

    plotModel1 = plot(tp1, p1, layout = (1, 2), legend=false, size=(1200, 620), margin=8mm,
        plot_title="Education Prediction using Age and Hours Worked per Week")
    annotate!(24, 1, [text("Fold " * string(k), 9)])
    savefig(plotModel1, "PlotModel_1f" * string(k) * ".png")
end

gif(anim, "PlotModel1.gif", fps=1)

println("x-hats: "); display(xhats1);
println("RMS errors: "); display(rmsErrors1);
```

```
x-hats:
[ Info: Saved animation to C:\Users\bento\OneDrive\School Documents\Spring 2023\MAS 4106\Final Project\PlotModel1.gif
3×5 Matrix{Float64}:
 12.8994      12.8686      12.9264      12.8496      12.8789
  0.00430604   0.00428053   0.00302945   0.00392448   0.00370823
  0.0274823    0.0279695    0.0275296    0.0287255    0.0282558
RMS errors:
2×5 Matrix{Float64}:
 2.6674   2.66501  2.68687  2.69017  2.67073
 2.71069  2.71993  2.63268  2.61902  2.69737
```

Model 2: Work Class & Occupation

In [7]:
```
# Create the matrix for model 2
Amod2 = [ones(m) workClassMatrix occMatrix]

# Create a number storing the number of elements in each fold
fold = div(m,5);

# Create a random permutation of m-values
I = Random.randperm(m);

# Storage for cross-validation results
xhats2 = zeros(8,5);
rmsErrors2 = zeros(2,5);

# For each fold, compute the appropriate model and store its error.
anim = @animate for k = 1:5

    # Assign a random permutation into training and test data
    if (k == 1)
        Itest = I[1 : fold];
        Itrain = I[fold+1 : end];
    elseif (k == 5)
        Itest = I[4 * fold+1 : end];
        Itrain = I[1:4 * fold];
    else
        Itest = I[(k-1) * fold+1: k*fold];
        Itrain = I[[1 : (k-1) * fold ; k * fold + 1 : m]];
    end

    #display(Itest);
    #display(Itrain);

    # Compute sample sizes for training and test data
    mTest = length(Itest);
    mTrain = length(Itrain);

    # Compute the model based on training data and store it
    xhats2[:, k] = Amod2[Itrain, :] \ eduClass[Itrain];

    # Compute RMS errors for both training and test data
    rmsErrors2[1,k] = norm(Amod2[Itrain,:] * xhats2[:, k] - eduClass[Itrain]) / sqrt(mTrain);
    rmsErrors2[2,k] = norm(Amod2[Itest,:] * xhats2[:, k] - eduClass[Itest]) / sqrt(mTest);
```

```
    ## Plot actual (x) versus predicted (y) education
    # Plot test data
    tp2 = scatter(eduClass[Itest], Amod2[Itest, :] * xhats2[:, k], color="firebrick1", xlims=(0,25), ylims=(0,25),
        title="Test data  " * L"(n=%$mTest)", xlabel="Actual education (years)",
        ylabel="Predicted education (years)", minorgrid=true)
    plot!([0,25], [0,25], linestyle = :dash, linewidth=2, color="magenta") # y=x line

    # Plot training data
    p2 = scatter(eduClass[Itrain], Amod2[Itrain, :] * xhats2[:, k], color="firebrick1", xlims=(0,25), ylims=(0,25),
        title="Training data  " * L"(n=%$mTrain)", xlabel="Actual education (years)",
        ylabel="Predicted education (years)", minorgrid=true)
    plot!([0,25], [0,25], linestyle = :dash, linewidth=2, color="magenta") # y=x line

    plotModel2 = plot(tp2, p2, layout = (1, 2), legend=false, size=(1200, 620), margin=8mm,
        plot_title="Education Prediction using Work Class and Occupation")
    annotate!(24, 1, [text("Fold " * string(k), 9)])
    savefig(plotModel2, "PlotModel_2f" * string(k) * ".png")
end

gif(anim, "PlotModel2.gif", fps=1)

println("x-hats: "); display(xhats2);
println("RMS errors: "); display(rmsErrors2);
```

```
x-hats:
[ Info: Saved animation to C:\Users\bento\OneDrive\School Documents\Spring 2023\MAS 4106\Final Project\PlotModel2.gif
8×5 Matrix{Float64}:
 12.3024    12.6801    12.5264    12.6603    12.7804
  0.660942   0.27953    0.463756   0.339932   0.213119
  0.966933   0.567723   0.791486   0.640428   0.529895
  1.85696    1.40346    1.60298    1.49303    1.36053
  0.25009    0.274662   0.264232   0.226239   0.255412
  1.55111    1.55888    1.54229    1.5181     1.53125
  1.79753    1.82916    1.81859    1.79032    1.77546
 -0.329087  -0.341501  -0.376461  -0.377015  -0.367817
RMS errors:
2×5 Matrix{Float64}:
 2.50131  2.49474  2.50813  2.50927  2.50833
 2.51717  2.54292  2.48982  2.4849   2.48874
```

Model 3: Marital Status & Native Country

In [8]:

```
# Create the matrix for model 3
Amod3 = [ones(m) marryMatrix natMatrix]

# Create a number storing the number of elements in each fold
fold = div(m,5);

# Create a random permutation of m-values
I = Random.randperm(m);

# Storage for cross-validation results
xhats3 = zeros(6,5);
rmsErrors3 = zeros(2,5);

# For each fold, compute the appropriate model and store its error.
anim = @animate for k = 1:5

    # Assign a random permutation into training and test data
    if (k == 1)
        Itest = I[1 : fold];
        Itrain = I[fold+1 : end];
    elseif (k == 5)
        Itest = I[4 * fold+1 : end];
        Itrain = I[1:4 * fold];
    else
        Itest = I[(k-1) * fold+1: k*fold];
        Itrain = I[[1 : (k-1) * fold ; k * fold + 1 : m]];
    end

    #display(Itest);
    #display(Itrain);

    # Compute sample sizes for training and test data
    mTest = length(Itest);
    mTrain = length(Itrain);

    # Compute the model based on training data and store it
    xhats3[:, k] = Amod3[Itrain, :] \ eduClass[Itrain];

    # Compute RMS errors for both training and test data
    rmsErrors3[1,k] = norm(Amod3[Itrain,:] * xhats3[:, k] - eduClass[Itrain]) / sqrt(mTrain);
    rmsErrors3[2,k] = norm(Amod3[Itest,:] * xhats3[:, k] - eduClass[Itest]) / sqrt(mTest);
```

```
    ## Plot actual (x) versus predicted (y) education
    # Plot test data
    tp3 = scatter(eduClass[Itest], Amod3[Itest, :] * xhats3[:, k], color="olive", xlims=(0,25), ylims=(0,25),
        title="Test data  " * L"(n=%$mTest)", xlabel="Actual education (years)",
        ylabel="Predicted education (years)", minorgrid=true)
    plot!([0,25], [0,25], linestyle = :dash, linewidth=2, color="magenta") # y=x line

    # Plot training data
    p3 = scatter(eduClass[Itrain], Amod3[Itrain, :] * xhats3[:, k], color="olive", xlims=(0,25), ylims=(0,25),
        title="Training data  " * L"(n=%$mTrain)", xlabel="Actual education (years)",
        ylabel="Predicted education (years)", minorgrid=true)
    plot!([0,25], [0,25], linestyle = :dash, linewidth=2, color="magenta") # y=x line

    plotModel3 = plot(tp3, p3, layout = (1, 2), legend=false, size=(1200, 620), margin=8mm,
        plot_title="Education Prediction using Marital Status and Native Country")
    annotate!(24, 1, [text("Fold " * string(k), 9)])
    savefig(plotModel3, "PlotModel_3f" * string(k) * ".png")
end

gif(anim, "PlotModel3.gif", fps=1)

println("x-hats: "); display(xhats3);
println("RMS errors: "); display(rmsErrors3);
```

```
x-hats:
[ Info: Saved animation to C:\Users\bento\OneDrive\School Documents\Spring 2023\MAS 4106\Final Project\PlotModel3.gif
6×5 Matrix{Float64}:
 13.9877     13.8908     13.947      13.7664    13.92
  0.155258    0.17962     0.178169    0.207358   0.191497
  0.403364    0.401006    0.385306    0.446079   0.38273
 -0.0589282   0.0167629  -0.0206474   0.13698   -0.0163766
 -1.90223    -1.97846    -1.73949    -1.49969   -2.0847
  0.975744    1.0419      1.10502     1.16905    0.951724
RMS errors:
2×5 Matrix{Float64}:
 2.68177  2.68777  2.68451  2.67507  2.69547
 2.69831  2.67427  2.68748  2.72551  2.64379
```

Model 4: Race & Sex

In [9]:

```
# Create the matrix for model 4
Amod4 = [ones(m) raceMatrix sexClass]

# Create a number storing the number of elements in each fold
fold = div(m,5);

# Create a random permutation of m-values
I = Random.randperm(m);

# Storage for cross-validation results
xhats4 = zeros(6,5);
rmsErrors4 = zeros(2,5);

# For each fold, compute the appropriate model and store its error.
anim = @animate for k = 1:5

    # Assign a random permutation into training and test data
    if (k == 1)
        Itest = I[1 : fold];
        Itrain = I[fold+1 : end];
    elseif (k == 5)
        Itest = I[4 * fold+1 : end];
        Itrain = I[1:4 * fold];
    else
        Itest = I[(k-1) * fold+1: k*fold];
        Itrain = I[[1 : (k-1) * fold ; k * fold + 1 : m]];
    end

    #display(Itest);
    #display(Itrain);

    # Compute sample sizes for training and test data
    mTest = length(Itest);
    mTrain = length(Itrain);

    # Compute the model based on training data and store it
    xhats4[:, k] = Amod4[Itrain, :] \ eduClass[Itrain];

    # Compute RMS errors for both training and test data
    rmsErrors4[1,k] = norm(Amod4[Itrain,:] * xhats4[:, k] - eduClass[Itrain]) / sqrt(mTrain);
    rmsErrors4[2,k] = norm(Amod4[Itest,:] * xhats4[:, k] - eduClass[Itest]) / sqrt(mTest);
```

```julia
    ## Plot actual (x) versus predicted (y) education
    # Plot test data
    tp4 = scatter(eduClass[Itest], Amod4[Itest, :] * xhats4[:, k], color="peru", xlims=(0,25), ylims=(0,25),
        title="Test data  " * L"(n=%$mTest)", xlabel="Actual education (years)",
        ylabel="Predicted education (years)", minorgrid=true)
    plot!([0,25], [0,25], linestyle = :dash, linewidth=2, color="magenta") # y=x line

    # Plot training data
    p4 = scatter(eduClass[Itrain], Amod4[Itrain, :] * xhats4[:, k], color="peru", xlims=(0,25), ylims=(0,25),
        title="Training data  " * L"(n=%$mTrain)", xlabel="Actual education (years)",
        ylabel="Predicted education (years)", minorgrid=true)
    plot!([0,25], [0,25], linestyle = :dash, linewidth=2, color="magenta") # y=x line

    plotModel4 = plot(tp4, p4, layout = (1, 2), legend=false, size=(1200, 620), margin=8mm,
        plot_title="Education Prediction using Race and Sex")
    annotate!(24, 1, [text("Fold " * string(k), 9)])
    savefig(plotModel4, "PlotModel_4f" * string(k) * ".png")
end

gif(anim, "PlotModel4.gif", fps=1)

println("x-hats: "); display(xhats4);
println("RMS errors: "); display(rmsErrors4);
```

```
x-hats:
[ Info: Saved animation to C:\Users\bento\OneDrive\School Documents\Spring 2023\MAS 4106\Final Project\PlotModel4.gif
6×5 Matrix{Float64}:
 12.6543     12.808      12.5665     12.6829     12.6842
  1.56535     1.40948     1.64869     1.53534     1.52438
  2.36725     2.20714     2.55412     2.32903     2.33532
  0.754517    0.766639    0.880823    0.84268     0.819563
  0.935141    0.765791    1.00975     0.925382    0.939997
  0.0519982   0.0348162   0.0436678   0.0433617   0.0795852
RMS errors:
2×5 Matrix{Float64}:
 2.69214  2.66804  2.68784  2.69769  2.67477
 2.65212  2.74802  2.66976  2.62942  2.72176
```

Model 5: GDP + Income

In [10]:

```
# Create the matrix for model 5
Amod5 = [ones(m) gdpClass incomeClass]

# Create a number storing the number of elements in each fold
fold = div(m,5);

# Create a random permutation of m-values
I = Random.randperm(m);

# Storage for cross-validation results
xhats5 = zeros(3,5);
rmsErrors5 = zeros(2,5);

# For each fold, compute the appropriate model and store its error.
anim = @animate for k = 1:5

    # Assign a random permutation into training and test data
    if (k == 1)
        Itest = I[1 : fold];
        Itrain = I[fold+1 : end];
    elseif (k == 5)
        Itest = I[4 * fold+1 : end];
        Itrain = I[1:4 * fold];
    else
        Itest = I[(k-1) * fold+1: k*fold];
        Itrain = I[[1 : (k-1) * fold ; k * fold + 1 : m]];
    end

    #display(Itest);
    #display(Itrain);

    # Compute sample sizes for training and test data
    mTest = length(Itest);
    mTrain = length(Itrain);

    # Compute the model based on training data and store it
    xhats5[:, k] = Amod5[Itrain, :] \ eduClass[Itrain];

    # Compute RMS errors for both training and test data
    rmsErrors5[1,k] = norm(Amod5[Itrain,:] * xhats5[:, k] - eduClass[Itrain]) / sqrt(mTrain);
    rmsErrors5[2,k] = norm(Amod5[Itest,:] * xhats5[:, k] - eduClass[Itest]) / sqrt(mTest);
```

```
    ## Plot actual (x) versus predicted (y) education
    # Plot test data
    tp5 = scatter(eduClass[Itest], Amod5[Itest, :] * xhats5[:, k], color="purple1", xlims=(0,25), ylims=(0,25),
        title="Test data  " * L"(n=%$mTest)", xlabel="Actual education (years)",
        ylabel="Predicted education (years)", minorgrid=true)
    plot!([0,25], [0,25], linestyle = :dash, linewidth=2, color="magenta") # y=x line

    # Plot training data
    p5 = scatter(eduClass[Itrain], Amod5[Itrain, :] * xhats5[:, k], color="purple1", xlims=(0,25), ylims=(0,25),
        title="Training data  " * L"(n=%$mTrain)", xlabel="Actual education (years)",
        ylabel="Predicted education (years)", minorgrid=true)
    plot!([0,25], [0,25], linestyle = :dash, linewidth=2, color="magenta") # y=x line

    plotModel5 = plot(tp5, p5, layout = (1, 2), legend=false, size=(1200, 620), margin=8mm,
        plot_title="Education Prediction using GDP and Income")
    annotate!(24, 1, [text("Fold " * string(k), 9)])
    savefig(plotModel5, "PlotModel_5f" * string(k) * ".png")
end

gif(anim, "PlotModel5.gif", fps=1)

println("x-hats: "); display(xhats5);
println("RMS errors: "); display(rmsErrors5);
```

```
x-hats:
[ Info: Saved animation to C:\Users\bento\OneDrive\School Documents\Spring 2023\MAS 4106\Final Project\PlotModel5.gif
3×5 Matrix{Float64}:
 14.3296      14.3196      14.3803      14.2776      14.2658
  1.90156e-7   1.90884e-7   1.81882e-7   1.99817e-7   2.01494e-7
 -1.88648     -1.89267     -1.90266     -1.91815     -1.92987
RMS errors:
2×5 Matrix{Float64}:
 2.53206  2.54077  2.53352  2.5438   2.54818
 2.5704   2.5356   2.56484  2.52359  2.50611
```

Model 6: HDI & Income

In [11]:
```julia
# Create the matrix for model 6
Amod6 = [ones(m) marryMatrix natMatrix]

# Create a number storing the number of elements in each fold
fold = div(m,5);

# Create a random permutation of m-values
I = Random.randperm(m);

# Storage for cross-validation results
xhats6 = zeros(6,5);
rmsErrors6 = zeros(2,5);

# For each fold, compute the appropriate model and store its error.
anim = @animate for k = 1:5

    # Assign a random permutation into training and test data
    if (k == 1)
        Itest = I[1 : fold];
        Itrain = I[fold+1 : end];
    elseif (k == 5)
        Itest = I[4 * fold+1 : end];
        Itrain = I[1:4 * fold];
    else
        Itest = I[(k-1) * fold+1: k*fold];
        Itrain = I[[1 : (k-1) * fold ; k * fold + 1 : m]];
    end

    #display(Itest);
    #display(Itrain);

    # Compute sample sizes for training and test data
    mTest = length(Itest);
    mTrain = length(Itrain);

    # Compute the model based on training data and store it
    xhats6[:, k] = Amod6[Itrain, :] \ eduClass[Itrain];

    # Compute RMS errors for both training and test data
    rmsErrors6[1,k] = norm(Amod6[Itrain,:] * xhats6[:, k] - eduClass[Itrain]) / sqrt(mTrain);
    rmsErrors6[2,k] = norm(Amod6[Itest,:] * xhats6[:, k] - eduClass[Itest]) / sqrt(mTest);
```

```
    ## Plot actual (x) versus predicted (y) education
    # Plot test data
    tp6 = scatter(eduClass[Itest], Amod6[Itest, :] * xhats6[:, k], color="steelblue", xlims=(0,25), ylims=(0,25),
        title="Test data  " * L"(n=%$mTest)", xlabel="Actual education (years)",
        ylabel="Predicted education (years)", minorgrid=true)
    plot!([0,25], [0,25], linestyle = :dash, linewidth=2, color="magenta") # y=x line

    # Plot training data
    p6 = scatter(eduClass[Itrain], Amod6[Itrain, :] * xhats6[:, k], color="steelblue", xlims=(0,25), ylims=(0,25),
        title="Training data  " * L"(n=%$mTrain)", xlabel="Actual education (years)",
        ylabel="Predicted education (years)", minorgrid=true)
    plot!([0,25], [0,25], linestyle = :dash, linewidth=2, color="magenta") # y=x line

    plotModel6 = plot(tp6, p6, layout = (1, 2), legend=false, size=(1200, 620), margin=8mm,
        plot_title="Education Prediction using HDI and Income")
    annotate!(24, 1, [text("Fold " * string(k), 9)])
    savefig(plotModel6, "PlotModel_6f" * string(k) * ".png")
end

gif(anim, "PlotModel6.gif", fps=1)

println("x-hats: "); display(xhats6);
println("RMS errors: "); display(rmsErrors6);
```

```
x-hats:
[ Info: Saved animation to C:\Users\bento\OneDrive\School Documents\Spring 2023\MAS 4106\Final Project\PlotModel6.gif
6×5 Matrix{Float64}:
 13.8278     13.9195      14.0067     13.8228     13.9365
  0.196929    0.187367     0.158714    0.207468    0.161924
  0.425387    0.405914     0.407843    0.412412    0.366848
  0.0755543  -0.00672327  -0.0846035   0.0820998  -0.0103343
 -1.60231    -1.85574     -1.92434    -1.92336    -1.8986
  1.20561     0.961067     0.960396    1.06467     1.04931
RMS errors:
2×5 Matrix{Float64}:
 2.68928  2.68035  2.68028  2.68773  2.68721
 2.66851  2.70382  2.70428  2.67458  2.67664
```

Education Prediction Models Factoring Controllable and Uncontrollable Attributes

Model 7: Education Prediction using Uncontrollable Attributes

In [12]:

```
# Create the matrix for model 7
Amod7 = [ones(m) natMatrix gdpClass./1000 raceMatrix sexClass ageClass]

# Create a number storing the number of elements in each fold
fold = div(m,5);

# Create a random permutation of m-values
I = Random.randperm(m);

# Storage for cross-validation results
xhats7 = zeros(11,5);
rmsErrors7 = zeros(2,5);

# For each fold, compute the appropriate model and store its error.
anim = @animate for k = 1:5

    # Assign a random permutation into training and test data
    if (k == 1)
        Itest = I[1 : fold];
        Itrain = I[fold+1 : end];
    elseif (k == 5)
        Itest = I[4 * fold+1 : end];
        Itrain = I[1:4 * fold];
    else
        Itest = I[(k-1) * fold+1: k*fold];
        Itrain = I[[1 : (k-1) * fold ; k * fold + 1 : m]];
    end

    #display(Itest);
    #display(Itrain);

    # Compute sample sizes for training and test data
    mTest = length(Itest);
    mTrain = length(Itrain);

    # Compute the model based on training data and store it
    xhats7[:, k] = Amod7[Itrain, :] \ eduClass[Itrain];

    # Compute RMS errors for both training and test data
    rmsErrors7[1,k] = norm(Amod7[Itrain,:] * xhats7[:, k] - eduClass[Itrain]) / sqrt(mTrain);
    rmsErrors7[2,k] = norm(Amod7[Itest,:] * xhats7[:, k] - eduClass[Itest]) / sqrt(mTest);
```

```
    ## Plot actual (x) versus predicted (y) education
    # Plot test data
    tp7 = scatter(eduClass[Itest], Amod7[Itest, :] * xhats7[:, k], color="green1", xlims=(0,25), ylims=(0,25),
        title="Test data  " * L"(n=%$mTest)", xlabel="Actual education (years)",
        ylabel="Predicted education (years)", minorgrid=true)
    plot!([0,25], [0,25], linestyle = :dash, linewidth=2, color="magenta") # y=x line

    # Plot training data
    p7 = scatter(eduClass[Itrain], Amod7[Itrain, :] * xhats7[:, k], color="green1", xlims=(0,25), ylims=(0,25),
        title="Training data  " * L"(n=%$mTrain)", xlabel="Actual education (years)",
        ylabel="Predicted education (years)", minorgrid=true)
    plot!([0,25], [0,25], linestyle = :dash, linewidth=2, color="magenta") # y=x line

    plotModel7 = plot(tp7, p7, layout = (1, 2), legend=false, size=(1200, 620), margin=8mm,
        plot_title="Education Prediction using Uncontrollable Attributes")
    annotate!(24, 1, [text("Fold " * string(k), 9)])
    savefig(plotModel7, "PlotModel_7f" * string(k) * ".png")
end

gif(anim, "PlotModel7.gif", fps=1)

println("x-hats: "); display(xhats7);
println("RMS errors: "); display(rmsErrors7);
```

```
x-hats:
[ Info: Saved animation to C:\Users\bento\OneDrive\School Documents\Spring 2023\MAS 4106\Final Project\PlotModel7.gif
11×5 Matrix{Float64}:
 12.9912       12.8312       13.006        12.9797       12.9836
 -2.71154      -2.65146      -2.66351      -2.65197      -2.62748
 -1.47366      -1.19728      -1.19496      -1.5245       -1.39079
  0.808007      1.06584       1.04648       0.92011       0.97633
  0.000447086   0.000457191   0.000457476   0.000452137   0.000440715
  0.649516      0.652681      0.481505      0.511816      0.569707
  1.01588       1.07143       0.790698      0.810964      0.856875
 -0.0833138    -0.0856961    -0.245154     -0.232276     -0.265763
  0.0408596     0.0722858    -0.0975334    -0.0493169    -0.0360692
  0.0414664     0.0307907     0.0418733     0.0686866     0.054818
  0.00380086    0.00471161    0.00473928    0.00511425    0.0050223
RMS errors:
2×5 Matrix{Float64}:
 2.60677  2.60663  2.62095  2.6162   2.60556
 2.62991  2.6307   2.57272  2.59231  2.63457
```

Model 8: Education Prediction using Controllable Attributes

In [13]:

```
# Create the matrix for model 8
Amod8 = [ones(m) workClassMatrix occMatrix marryMatrix incomeClass hrsPerWeekClass]

# Create a number storing the number of elements in each fold
fold = div(m,5);

# Create a random permutation of m-values
I = Random.randperm(m);

# Storage for cross-validation results
xhats8 = zeros(12,5);
rmsErrors8 = zeros(2,5);

# For each fold, compute the appropriate model and store its error.
anim = @animate for k = 1:5

    # Assign a random permutation into training and test data
    if (k == 1)
        Itest = I[1 : fold];
        Itrain = I[fold+1 : end];
    elseif (k == 5)
        Itest = I[4 * fold+1 : end];
        Itrain = I[1:4 * fold];
    else
        Itest = I[(k-1) * fold+1: k*fold];
        Itrain = I[[1 : (k-1) * fold ; k * fold + 1 : m]];
    end

    #display(Itest);
    #display(Itrain);

    # Compute sample sizes for training and test data
    mTest = length(Itest);
    mTrain = length(Itrain);

    # Compute the model based on training data and store it
    xhats8[:, k] = Amod8[Itrain, :] \ eduClass[Itrain];

    # Compute RMS errors for both training and test data
    rmsErrors8[1,k] = norm(Amod8[Itrain,:] * xhats8[:, k] - eduClass[Itrain]) / sqrt(mTrain);
    rmsErrors8[2,k] = norm(Amod8[Itest,:] * xhats8[:, k] - eduClass[Itest]) / sqrt(mTest);
```

```julia
    ## Plot actual (x) versus predicted (y) education
    # Plot test data
    tp8 = scatter(eduClass[Itest], Amod8[Itest, :] * xhats8[:, k], color="darkturquoise", xlims=(0,25), ylims=(0,25),
        title="Test data  " * L"(n=%$mTest)", xlabel="Actual education (years)",
        ylabel="Predicted education (years)", minorgrid=true)
    plot!([0,25], [0,25], linestyle = :dash, linewidth=2, color="magenta") # y=x line

    # Plot training data
    p8 = scatter(eduClass[Itrain], Amod8[Itrain, :] * xhats8[:, k], color="darkturquoise", xlims=(0,25), ylims=(0,25),
        title="Training data  " * L"(n=%$mTrain)", xlabel="Actual education (years)",
        ylabel="Predicted education (years)", minorgrid=true)
    plot!([0,25], [0,25], linestyle = :dash, linewidth=2, color="magenta") # y=x line

    plotModel8 = plot(tp8, p8, layout = (1, 2), legend=false, size=(1200, 620), margin=8mm,
        plot_title="Education Prediction using Controllable Attributes")
    annotate!(24, 1, [text("Fold " * string(k), 9)])
    savefig(plotModel8, "PlotModel_8f" * string(k) * ".png")

end

gif(anim, "PlotModel8.gif", fps=1)

println("x-hats: "); display(xhats8);
println("RMS errors: "); display(rmsErrors8);
```

```
x-hats:
[ Info: Saved animation to C:\Users\bento\OneDrive\School Documents\Spring 2023\MAS 4106\Final Project\PlotModel8.gif
12×5 Matrix{Float64}:
 13.4022     13.9564     13.583      13.6111     13.4468
  0.301582   -0.195846    0.0752428   0.0932641   0.217653
  0.490914   -0.0220096   0.275397    0.266489    0.392986
  1.35251     0.858254    1.19133     1.19688     1.32924
  0.321861    0.304129    0.318177    0.393806    0.369938
  1.38233     1.37111     1.41532     1.43398     1.4349
  1.66814     1.62716     1.68084     1.68208     1.67331
 -0.195334   -0.173393   -0.140361   -0.16566    -0.125013
  0.477888    0.465173    0.455267    0.437697    0.457018
 -0.313996   -0.303767   -0.31001    -0.307159   -0.307739
 -1.72593    -1.71854    -1.70681    -1.70669    -1.71073
  0.0158297   0.0147896   0.0159237   0.0148424   0.0151352
RMS errors:
```

```
2×5 Matrix{Float64}:
 2.40803  2.39555  2.38736  2.41019  2.40851
 2.3784   2.42818  2.46019  2.36961  2.37632
```

Human Development Index Prediction Models Factoring Controllable and Uncontrollable Attributes

Model 9: HDI Prediction using Uncontrollable Attributes

In [14]:

```
# Create the matrix for model 9
Amod9 = [ones(m) natMatrix gdpClass./1000 raceMatrix sexClass ageClass]

# Create a number storing the number of elements in each fold
fold = div(m,5);

# Create a random permutation of m-values
I = Random.randperm(m);

# Storage for cross-validation results
xhats9 = zeros(11,5);
rmsErrors9 = zeros(2,5);

# For each fold, compute the appropriate model and store its error.
anim = @animate for k = 1:5

    # Assign a random permutation into training and test data
    if (k == 1)
        Itest = I[1 : fold];
        Itrain = I[fold+1 : end];
    elseif (k == 5)
        Itest = I[4 * fold+1 : end];
        Itrain = I[1:4 * fold];
    else
        Itest = I[(k-1) * fold+1: k*fold];
        Itrain = I[[1 : (k-1) * fold ; k * fold + 1 : m]];
    end

    #display(Itest);
    #display(Itrain);

    # Compute sample sizes for training and test data
    mTest = length(Itest);
    mTrain = length(Itrain);

    # Compute the model based on training data and store it
    xhats9[:, k] = Amod9[Itrain, :] \ hdiClass[Itrain];

    # Compute RMS errors for both training and test data
    rmsErrors9[1,k] = norm(Amod9[Itrain,:] * xhats9[:, k] - hdiClass[Itrain]) / sqrt(mTrain);
    rmsErrors9[2,k] = norm(Amod9[Itest,:] * xhats9[:, k] - hdiClass[Itest]) / sqrt(mTest);
```

```julia
    ## Plot actual (x) versus predicted (y) HDI
    # Plot test data
    tp9 = scatter(hdiClass[Itest], Amod9[Itest, :] * xhats9[:, k], color="tomato", xlims=(0,1), ylims=(0,1),
        title="Test data  " * L"(n=%$mTest)", xlabel="Actual HDI",
        ylabel="Predicted HDI", minorgrid=true)
    plot!([0,1], [0,1], linestyle = :dash, linewidth=2, color="magenta") # y=x line

    # Plot training data
    p9 = scatter(hdiClass[Itrain], Amod9[Itrain, :] * xhats9[:, k], color="tomato", xlims=(0,1), ylims=(0,1),
        title="Training data  " * L"(n=%$mTrain)", xlabel="Actual HDI",
        ylabel="Predicted HDI", minorgrid=true)
    plot!([0,1], [0,1], linestyle = :dash, linewidth=2, color="magenta") # y=x line

    plotModel9 = plot(tp9, p9, layout = (1, 2), legend=false, size=(1200, 620), margin=8mm,
        plot_title="HDI Prediction using Uncontrollable Attributes")
    annotate!(0.96, 0.04, [text("Fold " * string(k), 9)])
    savefig(plotModel9, "PlotModel_9f" * string(k) * ".png")

end

gif(anim, "PlotModel9.gif", fps=1)

println("x-hats: "); display(xhats9);
println("RMS errors: "); display(rmsErrors9);
```

```
x-hats:
[ Info: Saved animation to C:\Users\bento\OneDrive\School Documents\Spring 2023\MAS 4106\Final Project\PlotModel9.gif
11×5 Matrix{Float64}:
  0.732739      0.732324     0.735522      0.730427      0.733534
 -0.108196     -0.107495    -0.108936     -0.106574     -0.107053
 -0.164898     -0.163813    -0.164533     -0.161627     -0.163752
 -0.177294     -0.175373    -0.176459     -0.172211     -0.1748
  3.31873e-5    3.2996e-5    3.32238e-5    3.29442e-5    3.30122e-5
  0.0047986     0.00599332   0.00258913    0.00703875    0.00399316
 -0.00300657   -0.0042954   -0.00640212   -0.00273069   -0.00673343
  0.00571047    0.00708785   0.00400962    0.00834088    0.004266
  0.000437104   0.00190798  -0.00143597    0.00336597   -0.000192443
  0.000683247   0.00108184   0.000608734   0.000952723   0.00107323
  3.5966e-5     3.06324e-5   3.29207e-5    3.8273e-5     3.71789e-5
RMS errors:
```

```
2×5 Matrix{Float64}:
 0.0245519  0.0245313  0.0242698  0.0242344  0.0245718
 0.0239702  0.0240478  0.0250922  0.025243   0.0238836
```

Model 10: HDI Prediction using Uncontrollable Attributes

In [15]:

```
# Create the matrix for model 10
Amod10 = [ones(m) workClassMatrix occMatrix marryMatrix incomeClass hrsPerWeekClass]

# Create a number storing the number of elements in each fold
fold = div(m,5);

# Create a random permutation of m-values
I = Random.randperm(m);

# Storage for cross-validation results
xhats10 = zeros(12,5);
rmsErrors10 = zeros(2,5);

# For each fold, compute the appropriate model and store its error.
anim = @animate for k = 1:5

    # Assign a random permutation into training and test data
    if (k == 1)
        Itest = I[1 : fold];
        Itrain = I[fold+1 : end];
    elseif (k == 5)
        Itest = I[4 * fold+1 : end];
        Itrain = I[1:4 * fold];
    else
        Itest = I[(k-1) * fold+1: k*fold];
        Itrain = I[[1 : (k-1) * fold ; k * fold + 1 : m]];
    end

    #display(Itest);
    #display(Itrain);

    # Compute sample sizes for training and test data
    mTest = length(Itest);
    mTrain = length(Itrain);

    # Compute the model based on training data and store it
    xhats10[:, k] = Amod10[Itrain, :] \ hdiClass[Itrain];

    # Compute RMS errors for both training and test data
    rmsErrors10[1,k] = norm(Amod10[Itrain,:] * xhats10[:, k] - hdiClass[Itrain]) / sqrt(mTrain);
    rmsErrors10[2,k] = norm(Amod10[Itest,:] * xhats10[:, k] - hdiClass[Itest]) / sqrt(mTest);
```

```julia
    ## Plot actual (x) versus predicted (y) HDI
    # Plot test data
    tp10 = scatter(hdiClass[Itest], Amod10[Itest, :] * xhats10[:, k], color="goldenrod", xlims=(0,1), ylims=(0,1),
        title="Test data  " * L"(n=%$mTest)", xlabel="Actual HDI",
        ylabel="Predicted HDI", minorgrid=true)
    plot!([0,1], [0,1], linestyle = :dash, linewidth=2, color="magenta") # y=x line

    # Plot training data
    p10 = scatter(hdiClass[Itrain], Amod10[Itrain, :] * xhats10[:, k], color="goldenrod", xlims=(0,1), ylims=(0,1),
        title="Training data  " * L"(n=%$mTrain)", xlabel="Actual HDI",
        ylabel="Predicted HDI", minorgrid=true)
    plot!([0,1], [0,1], linestyle = :dash, linewidth=2, color="magenta") # y=x line

    plotModel10 = plot(tp10, p10, layout = (1, 2), legend=false, size=(1200, 620), margin=8mm,
        plot_title="HDI Prediction using Controllable Attributes")
    annotate!(24/25, 1/25, [text("Fold " * string(k), 9)])
    savefig(plotModel10, "PlotModel_10f" * string(k) * ".png")

end

gif(anim, "PlotModel10.gif", fps=1)

println("x-hats: "); display(xhats10);
println("RMS errors: "); display(rmsErrors10);
```

```
x-hats:
[ Info: Saved animation to C:\Users\bento\OneDrive\School Documents\Spring 2023\MAS 4106\Final Project\PlotModel10.gif
12×5 Matrix{Float64}:
  0.880907     0.862141     0.85545      0.867331     0.860578
 -0.0256334   -0.0101176   -0.00190489  -0.0126536   -0.0103278
 -0.0204854   -0.0042137    0.00337651  -0.00712204  -0.00467248
 -0.0173051   -0.00131956   0.00710312  -0.00430605  -0.00202818
  0.00197489   0.00316758   0.00286457   0.00232467   0.00509916
  0.0128369    0.0136869    0.0138093    0.0123049    0.017115
  0.00814684   0.00927342   0.00806057   0.00753494   0.0112216
  0.00375515   0.00418064   0.00419216   0.00333075   0.00711527
 -0.00498494  -0.00537207  -0.00540521  -0.00479151  -0.00497631
 -0.0115268   -0.0116592   -0.0111199   -0.0111126   -0.0116144
 -0.00811335  -0.00685062  -0.00806474  -0.00844847  -0.00772548
  4.84579e-5   7.86322e-5   6.91479e-5   6.34919e-5   6.63228e-5
RMS errors:
```

```
2×5 Matrix{Float64}:
 0.0695176  0.0697388  0.0698516  0.0698121  0.0702932
 0.0711501  0.0702798  0.0698287  0.0699851  0.0680414
```

In [ ]: