Homework 2

Submitted on 3rd Oct, 2019 by

Aman Jain (aj644)

Question 1

Answer a

$$f(x) = log(\sum_{i}^{n} e^{x_i})$$
$$f(x) = log(e^{x_1} + e^{x_2} + ... + e^{x_n})$$
$$\frac{\partial f}{\partial x_1} = e^{x_1}/(e^{x_1} + e^{x_2} + ... + e^{x_n})$$
$$\frac{\partial f}{\partial x_2} = e^{x_2}/(e^{x_1} + e^{x_2} + ... + e^{x_n})$$
$$therfore,$$
$$\frac{\partial f}{\partial x_i} = e^{x_i}/(e^{x_1} + e^{x_2} + ... + e^{x_n})$$

Answer b

$$\nabla f(x) = \left[\frac{\partial f}{\partial x_1} \frac{\partial f}{\partial x_2} \frac{\partial f}{\partial x_3} .. \right]$$

$$\nabla f(x) = \left[\, e^{x_1}/(e^{x_1} + e^{x_2}+\ldots+e^{x_n}), e^{x_2}/(e^{x_1} + e^{x_2}+\ldots+e^{x_n}), e^{x_3}/(e^{x_1} + e^{x_2}+\ldots+e^{x_n}).\,.\right]$$

Answer c

Yes matrix chain rule exists. For matrix X and scalar f(x),

$$\frac{\mathrm{d}f(g(X)))}{\mathrm{d}x} = \frac{\mathrm{d}f(g(X)))}{\mathrm{d}g(X))} * \frac{\mathrm{d}(g(X))}{\mathrm{d}(X))}$$

Answer d and e

$$f(x) = X^T AX$$

Let X be n * 1 matrix. A will be n * n

$$X = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \end{bmatrix}$$

$$A = \begin{bmatrix} a11 & a12 & a13... & a1n \\ a21 & a22 & a23.... & a2n \\ . & . & . & . \\ an1 & an2 & an3.... & ann \end{bmatrix}$$

$$hence, f(x) = \left[\, x_1, x_2, x_3, .. \right] * \begin{bmatrix} a11 & a12 & a13... & a1n \\ a21 & a22 & a23.... & a2n \\ . & . & . & . \\ an1 & an2 & an3.... & ann \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \end{bmatrix}$$

$$hence, f(x) = \begin{bmatrix} x_1, x_2, x_3, .. \end{bmatrix} * \begin{bmatrix} \sum_1^n x_i * a_1 i \\ \sum_1^n x_i * a_2 i \\ . \\ . \\ \sum_1^n x_i * a_n i \end{bmatrix}$$

$$f(x) = \sum_1^n \sum_1^n a_i j x_i x_j$$

$$\frac{\partial f}{\partial x_i} = \sum_1^n (a_i j + a_j i) x_j$$

$$\triangledown f(x) = (A^T + A)X$$

Lets check it for n = 2,

$$hence, f(x) = \begin{bmatrix} x_1, x_2 \end{bmatrix} * \begin{bmatrix} a11 & a12 \\ a21 & a22 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$hence, f(x) = \begin{bmatrix} x_1, x_2 \end{bmatrix} * \begin{bmatrix} a11x_1 + a12x_2 \\ a21x_1 + a22x_2 \end{bmatrix}$$

$$hence, f(x) = a_{11}x_1x_1 + a_{12}x_1x_2 + a_{21}x_2x_1 + a_{22}x_2x_2$$

$$\frac{\partial f}{\partial x_1} = 2a_{11}x_1 + a_{12}x_2 + a_{21}x_2 = a_{11}x_1 + a_{12}x_2 + a_{21}x_2 + a_{11}x_1$$

$$\frac{\partial f}{\partial x_2} = 2a_{22}x_2 + a_{21}x_1 + a_{12}x_1 = a_{22}x_2 + a_{21}x_1 + a_{12}x_1 + a_{22}x_2$$

$$\triangledown f(x) = \begin{bmatrix} \frac{\partial}{\partial x_1} \\ \frac{\partial}{\partial x_2} \end{bmatrix}$$

$$\triangledown f(x) = (\begin{bmatrix} a11 + a12 \\ a21 + a22 \end{bmatrix} + \begin{bmatrix} a11 + a21 \\ a12 + a22 \end{bmatrix}) * \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\triangledown f(x) = (A^T + A)X$$

Answer f

Yes product rules exists in matrix form as well.

$$\nabla(fg) = f\nabla(g) + g\nabla(f)$$

Question 2

In [84]:
```
# generate values for testing
d = 5
X = randn(d)
v = randn(d)
```

Out[84]:
```
5-element Array{Float64,1}:
  0.6093688196204228
  0.8800129784687506
 -1.0135524142700199
  0.6446891634179082
  0.18201184518451033
```

In [85]:
```
using Plots
using LaTeXStrings
```

In [86]:
```
alphas = range(-1,stop=1,length=100)
```

Out[86]:
```
-1.0:0.020202020202020204:1.0
```

In [87]:
```
# Creating a function
function f(x)
    k = 0
    for i in range(1,size(X)[1])
        k = k + exp(x[i])
        #print("for i equals ",i," value considered is ",x[i], " and value of k is ",k, ".\n")
    end
    return log(k)
end
```

Out[87]: `f (generic function with 1 method)`

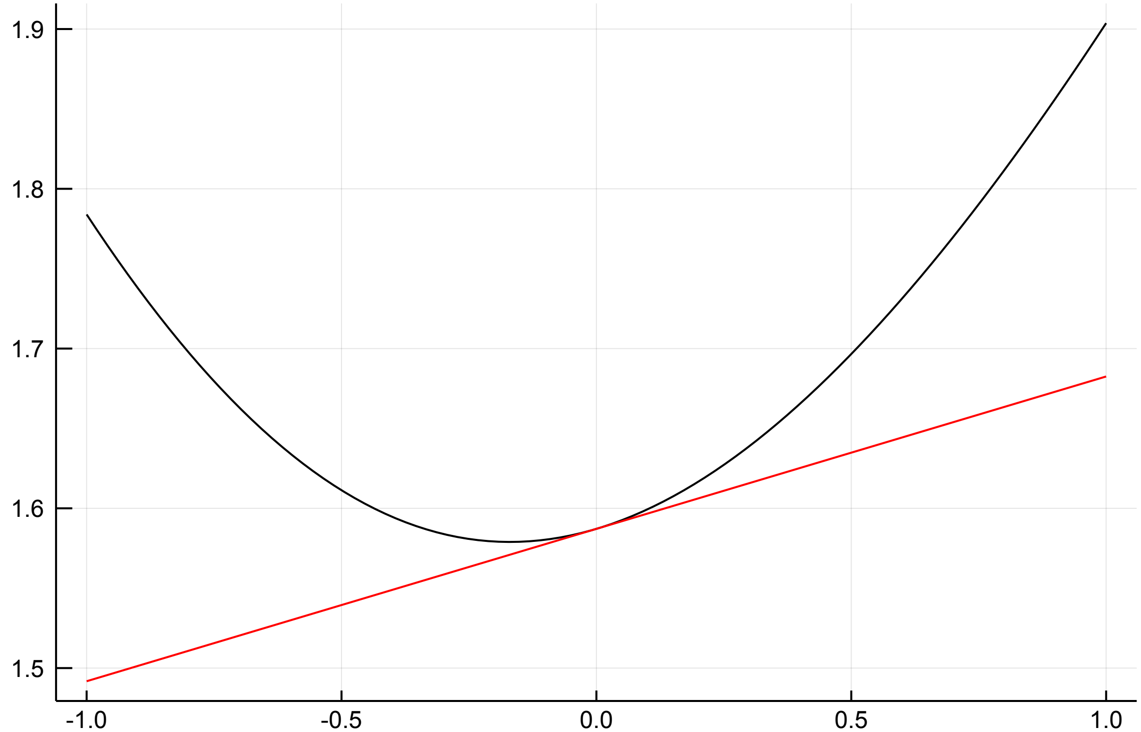
In [88]:
```
# Creating derivative of function
function ∇f(x)
    k = 0
    for i in range(1,size(X)[1])
        k = k + exp(x[i])
    end
    #print("final value of k is ",k, ".\n")

    y = zeros(d)
    #print("initial value of y is ",y, ".\n")

    for j in range(1,size(X)[1])
        y[j] = exp(x[j]) / k
        #print("for i equals ",i," value considered is ",x[i], " and value of k is ",k, ".\n")
    end

    return y

end
```

Out[88]: `∇f (generic function with 1 method)`

In [89]:
```
using LinearAlgebra
```

In [90]:
```
plot(alphas, [f(X + alpha*v) for alpha in alphas], color=:black, label=L"$f(X + \alpha v)$")
plot!(alphas, [f(X) + alpha*dot(∇f(X), v) for alpha in alphas], color=:red, label=L"$f(X) + \alpha (\nal
xlabel!(L"$\alpha$")
```

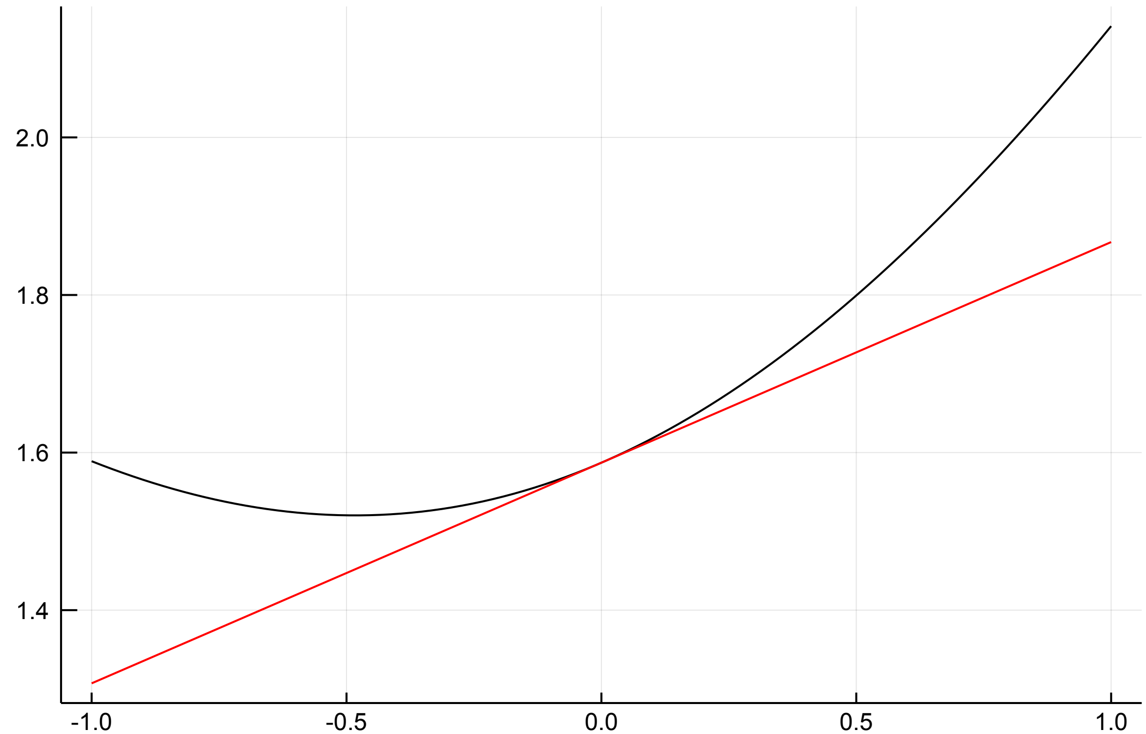
Out[90]:




```
sh: 1: dvipng: not found
dvipng: PNG conversion failed
sh: 1: dvipng: not found
dvipng: PNG conversion failed
sh: 1: dvipng: not found
dvipng: PNG conversion failed
```

In [91]:
```
v_2 = randn(d)
plot(alphas, [f(X + alpha*v_2) for alpha in alphas], color=:black, label=L"$f(X + \alpha v)$")
plot!(alphas, [f(X) + alpha*dot(∇f(X), v_2) for alpha in alphas], color=:red, label=L"$f(X) + \alpha (\n
xlabel!(L"$\alpha$")
```

Out[91]:

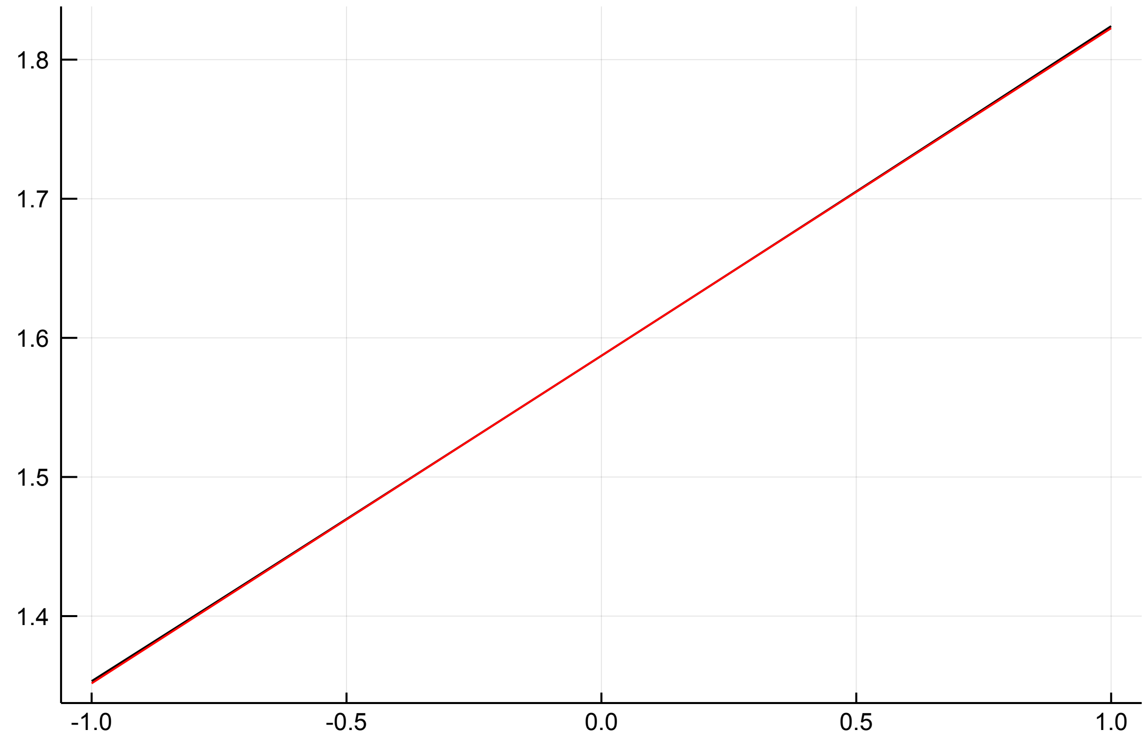



```
sh: 1: dvipng: not found
dvipng: PNG conversion failed
sh: 1: dvipng: not found
dvipng: PNG conversion failed
sh: 1: dvipng: not found
dvipng: PNG conversion failed
```

Answer 2(b)

In [92]:
```
v_2 = ∇f(X)
plot(alphas, [f(X + alpha*v_2) for alpha in alphas], color=:black, label=L"$f(X + \alpha v)$")
plot!(alphas, [f(X) + alpha*dot(∇f(X), v_2) for alpha in alphas], color=:red, label=L"$f(X) + \alpha (\n
xlabel!(L"$\alpha$")
```

Out[92]:




```
sh: 1: dvipng: not found
dvipng: PNG conversion failed
sh: 1: dvipng: not found
dvipng: PNG conversion failed
sh: 1: dvipng: not found
dvipng: PNG conversion failed
```

Answer 2(c)

function f decrease the fastest when v is in direction of delta f.

Question 3

$$\hat{y}_i(w) = x_i^T w$$

We say that a model with parameters w is accurate, on average, for the set of examples S ⊆ {1,...,n} if:

$$\sum_{i \subset S} (y_i - \hat{y}_i(w)) = 0$$

Answer a

Let's suppose that we are fitting a model with an offset: (xi)d, the last entry of xi, is equal to 1, for each i = 1,...,n. We will compute w using least squares, by **minimizing** with variable w ∈ Rd :

$$err(w) = \sum_{i=0}^{n} (y_i - x_i^T w)^2$$

differentiation wrt w we get,

$$err'(w) = 2 \sum_{i=0}^{n} (y_i - x_i^T w) * x_i^T$$

replacing y with predicted values:

$$err'(w) = 2 \sum_{i=0}^{n} (\hat{y}_i - x_i^T w) * x_i^T$$

but we have,

$$\hat{y}_i(w) = x_i^T w$$

hence,

$$err'(w) = 2 \sum_{i=0}^{n} (\hat{y}_i - \hat{y}_i) * x_i^T$$

$$err'(w) = 2\sum_{i=0}^{n}(0) * x_i^T$$
$$err'(w) = 0$$

hence the resulting linear model is accurate, on average, for the full set of examples S = {1, . . . , n}

Answer b

for X:
first entry of each feature vector is Boolean: (xi)1 ∊ {0, 1} for each i = 1, . . . , n.
(xi)d = 1 for each i = 1,...,n.
(xi)d = 1 for each i = 1,...,n.

$$\begin{bmatrix} 110 \\ \dots \\ \dots \\ 111 \end{bmatrix}$$

from answer a,

$$err'(w) = 2\sum_{i=0}^{n}(y_i - x_i^T w) * x_i^T$$
$$err'(w) = 2\sum_{i=0}^{n}(y_i - \hat{y}_i) * x_i^T$$

err'(w) = 0 As shown above the resulting linear model is accurate, on average. hence,

$$err'(w) = 2\sum_{i=0}^{n}(y_i - \hat{y}_i) * x_i^T = 0$$
$$(y_i - \hat{y}_i) * X^T = 0$$

in matrix notation.

$$(y_i - \hat{y}_i) * \begin{bmatrix} 110 \\ \cdots \\ \cdots \\ 111 \end{bmatrix} = 0$$

$$or \sum_{i \subset S_1} (y_i - \hat{y}) = 0$$

hence, the resulting linear model is accurate, on average, for the set of examples S1 = {i : (xi)1 = 1} for which the Boolean attribute has value 1.

Answer c

As shown in answer a,

$$\sum_{i \subset S} (y_i - \hat{y}_i(w)) = 0$$

and as proved in answer b,

$$\sum_{i \subset S_1} (y_i - \hat{y}_i(w)) = 0$$

please note that S1 = {i : (xi)1 = 1} and S0 = {i : (xi)1 = 0} are complementary set for S as (xi)1 is a boolean.

Substracting equation 2 from equation 1, we get,

$$\sum_{i \subset S_0} (y_i - \hat{y}_i(w)) = 0$$

Answer d

The input xi for each student is a vector of covariates representing we know about the student; for example,
one of the entries is a Boolean which takes value 1 if the student enjoyed homework 2, and 0 if the student hated homework 2;

other covariates might include the student's grades on each previous homework assignment,
the number of lectures the student attended,
the number of times the student went to office hours and to section,
the list of related classes the student had taken previously, etc.,

all encoded using a variety of feature transformations into the numerical vector xi.
The output y is the student's score on the final exam.

We build a model using least squares to predict the student's score from these covariates. Is the model accurate, on average?

Yes as proved in answer a, least square model will be accurate on average.

Is it accurate on average for students who enjoyed homework 2?

Yes as proved in answer b, least square model will be accurate for students who enjoyed homework 2 (hence boolean value of 1).

Is it accurate on average for students who hated homework 2?

Yes as proved in answer c, least square model will be accurate for students who hated homework 2 (hence boolean value of 0).

Answer e

Group 1: enjoyed homework 1.

Group 2: enjoyed homework 2.

Predicted score of Group 1 is higher than predicted score of Group 2, on average. As prediction error, on avergae, is 0 for both groups (Asnwer d), difference in predicted score should follow same trend as difference in the actuals for both groups.

Question 4

Answer a

In [93]:
```
# Install packages
Pkg.add("DataFrames")
Pkg.add("Plots")
Pkg.add("StatsPlots")
```

```
 Resolving package versions...
  Updating `~/.julia/Project.toml`
 [no changes]
  Updating `~/.julia/Manifest.toml`
 [no changes]
 Resolving package versions...
  Updating `~/.julia/Project.toml`
 [no changes]
  Updating `~/.julia/Manifest.toml`
 [no changes]
 Resolving package versions...
  Updating `~/.julia/Project.toml`
 [no changes]
  Updating `~/.julia/Manifest.toml`
 [no changes]
```

In [94]:
```
# bring packages into main namespace
using DataFrames             # Data tables are called "DataFrames"
using Plots, StatsPlots      # load plotting packages
using Statistics             # basic statistical functions
```

```
In [95]: # load data
tax = readtable("incomeTaxData.csv")
```

Out[95]: 12,589 rows × 13 columns

	Year	County	IncomeClass	Disclosure	ReturnCount	TotalIncome	TotalDeductions	TotalExemptions	TotalTaxableIncome	TotalT
	Int64⍰	String⍰	String⍰	String⍰	Int64⍰	Int64⍰	Int64⍰	Int64⍰	Int64⍰	
1	2011	Hamilton	500,000 and over	d/	missing	missing	missing	missing	missing	
2	2003	Dutchess	100,000 - 199,999	missing	12738	1668991	261775	16628	1390589	
3	2001	Ontario	Total	missing	44898	1829734	449962	25393	1354378	
4	2012	New York City - Richmond	30,000 - 39,999	missing	16086	559918	192402	10519	356997	
5	2007	Clinton	50,000 - 59,999	missing	2197	120482	29429	1649	89404	
6	1999	Saratoga	Total	missing	88920	3969074	883753	51293	3034030	
7	2010	Onondaga	60,000 - 74,999	missing	13837	928570	195635	11010	721925	
8	2008	Ulster	40,000 - 49,999	missing	5971	266988	77044	3435	186509	
9	2005	Clinton	Total	missing	33634	1284580	332270	17436	934774	
10	2007	Livingston	200,000 - 249,999	missing	104	22811	2165	135	20511	
11	2012	Washington	500,000 and over	missing	35	44397	1770	12	42614	
12	2011	New York City - Kings	100,000 - 199,999	missing	65072	8717982	1220269	57409	7440304	
13	2005	Westchester	50,000 - 59,999	missing	23546	1290957	352369	14290	924254	
14	2004	Orange	60,000 - 74,999	missing	11545	776872	193880	12641	570351	
15	2008	Dutchess	75,000 - 99,999	missing	12391	1077050	243939	12581	820530	

	Year	County	IncomeClass	Disclosure	ReturnCount	TotalIncome	TotalDeductions	TotalExemptions	TotalTaxableIncome	TotalT
	Int64⍰	String⍰	String⍰	String⍰	Int64⍰	Int64⍰	Int64⍰	Int64⍰	Int64⍰	
16	2012	Cortland	30,000 - 39,999	missing	2069	71838	22572	1308	47958	
17	2000	New York City - Kings	5,000 - 9,999	missing	125794	951643	885145	1297	65177	
18	2008	Dutchess	60,000 - 74,999	missing	9577	643643	162693	7664	473287	
19	2012	Ulster	5,000 - 9,999	missing	7048	52553	44434	68	8051	
20	2010	Ontario	200,000 - 249,999	missing	408	90828	9708	535	80585	
21	2006	Oswego	100,000 - 199,999	missing	2778	346108	49078	3367	293655	
22	2010	Oswego	Total	missing	50958	2023764	519529	28462	1492712	
23	2002	Sullivan	50,000 - 59,999	missing	1676	91734	22835	1638	67261	
24	2010	Rockland	10,000 - 19,999	missing	18173	268331	182753	7661	77918	
25	2003	Delaware	60,000 - 74,999	missing	1075	71844	15672	1093	55079	
26	2005	Washington	75,000 - 99,999	missing	1572	135013	24832	1705	108477	
27	2002	Fulton	100,000 - 199,999	missing	632	80160	10662	659	68840	
28	2006	Schoharie	20,000 - 29,999	missing	1769	43849	18874	1040	23935	
29	2003	Genesee	40,000 - 49,999	missing	2154	96606	27572	2161	66873	
30	2007	Seneca	40,000 - 49,999	missing	1174	52525	14772	902	36851	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	

In [96]:
```
# clean data
tax = tax[.!(ismissing.(tax[:ReturnCount])), :];
sort(tax, cols = :Year)

# create new columns
tax[:avg_tax] = tax[:TotalTaxLiability]./tax[:ReturnCount];
```

In [97]:
```
tax
```

Out[97]: 12,528 rows × 14 columns

	Year	County	IncomeClass	Disclosure	ReturnCount	TotalIncome	TotalDeductions	TotalExemptions	TotalTaxableIncome	TotalT
	Int64⍰	String⍰	String⍰	String⍰	Int64⍰	Int64⍰	Int64⍰	Int64⍰	Int64⍰	
1	2003	Dutchess	100,000 - 199,999	missing	12738	1668991	261775	16628	1390589	
2	2001	Ontario	Total	missing	44898	1829734	449962	25393	1354378	
3	2012	New York City - Richmond	30,000 - 39,999	missing	16086	559918	192402	10519	356997	
4	2007	Clinton	50,000 - 59,999	missing	2197	120482	29429	1649	89404	
5	1999	Saratoga	Total	missing	88920	3969074	883753	51293	3034030	
6	2010	Onondaga	60,000 - 74,999	missing	13837	928570	195635	11010	721925	
7	2008	Ulster	40,000 - 49,999	missing	5971	266988	77044	3435	186509	
8	2005	Clinton	Total	missing	33634	1284580	332270	17436	934774	
9	2007	Livingston	200,000 - 249,999	missing	104	22811	2165	135	20511	
10	2012	Washington	500,000 and over	missing	35	44397	1770	12	42614	
11	2011	New York City - Kings	100,000 - 199,999	missing	65072	8717982	1220269	57409	7440304	
12	2005	Westchester	50,000 - 59,999	missing	23546	1290957	352369	14290	924254	
13	2004	Orange	60,000 - 74,999	missing	11545	776872	193880	12641	570351	
14	2008	Dutchess	75,000 - 99,999	missing	12391	1077050	243939	12581	820530	
15	2012	Cortland	30,000 - 39,999	missing	2069	71838	22572	1308	47958	
16	2000	New York City - Kings	5,000 - 9,999	missing	125794	951643	885145	1297	65177	

	Year	County	IncomeClass	Disclosure	ReturnCount	TotalIncome	TotalDeductions	TotalExemptions	TotalTaxableIncome	TotalT
	Int64⍰	String⍰	String⍰	String⍰	Int64⍰	Int64⍰	Int64⍰	Int64⍰	Int64⍰	
17	2008	Dutchess	60,000 - 74,999	missing	9577	643643	162693	7664	473287	
18	2012	Ulster	5,000 - 9,999	missing	7048	52553	44434	68	8051	
19	2010	Ontario	200,000 - 249,999	missing	408	90828	9708	535	80585	
20	2006	Oswego	100,000 - 199,999	missing	2778	346108	49078	3367	293655	
21	2010	Oswego	Total	missing	50958	2023764	519529	28462	1492712	
22	2002	Sullivan	50,000 - 59,999	missing	1676	91734	22835	1638	67261	
23	2010	Rockland	10,000 - 19,999	missing	18173	268331	182753	7661	77918	
24	2003	Delaware	60,000 - 74,999	missing	1075	71844	15672	1093	55079	
25	2005	Washington	75,000 - 99,999	missing	1572	135013	24832	1705	108477	
26	2002	Fulton	100,000 - 199,999	missing	632	80160	10662	659	68840	
27	2006	Schoharie	20,000 - 29,999	missing	1769	43849	18874	1040	23935	
28	2003	Genesee	40,000 - 49,999	missing	2154	96606	27572	2161	66873	
29	2007	Seneca	40,000 - 49,999	missing	1174	52525	14772	902	36851	
30	2002	Albany	10,000 - 19,999	missing	19684	292594	176319	6563	109712	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	

Plot the number of returns in Tompkins County over time. (you should draw the plot for each income class and ignore the rows with the class of 'Total'.)

In [98]:
```
Tompkins = tax[tax[:County] .== "Tompkins",:]
Tompkins = tax_Tompkins[tax_Tompkins[:IncomeClass].!="Total",:]
!(tax_Tompkins, [order(:Year), order(:TotalTaxableIncome)])
Tompkins_aggr_sum = by(tax_Tompkins, :Year, :ReturnCount => sum, :TotalIncome => sum, :TotalTaxLiability
:_Tompkins_aggr_sum = by(tax_Tompkins, [:Year], df -> sum(df[:, :ReturnCount]))
't(tax_Tompkins, cols = :Year, :TotalTaxableIncome)
:_Tompkins_aggr = aggregate(tax_Tompkins, :Year, sum)
:_Tompkins_aggr
```

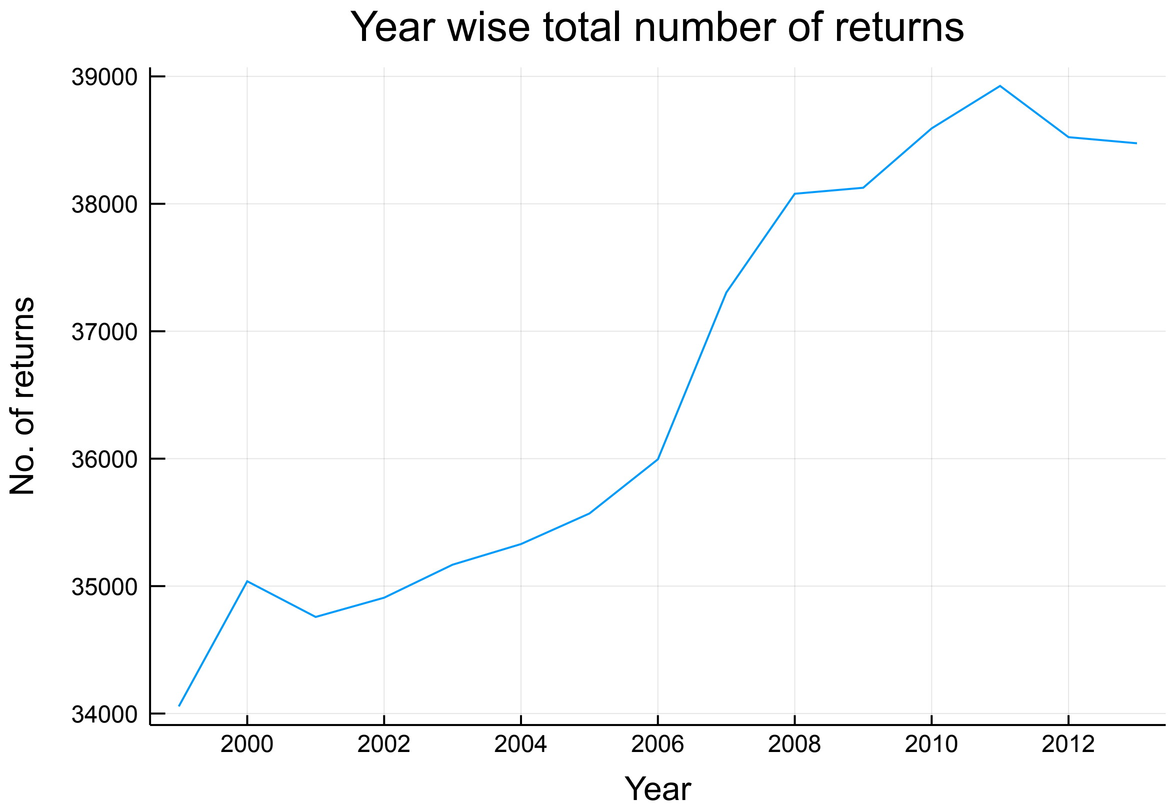
Out[98]: 15 rows × 5 columns

	Year Int64⍰	ReturnCount_sum Int64	TotalIncome_sum Int64	TotalTaxLiability_sum Int64	TotalExemptions_sum Int64
1	1999	34056	1420132	59378	17073
2	2000	35038	1585531	68997	17193
3	2001	34758	1493756	62816	16956
4	2002	34909	1476145	60372	16894
5	2003	35168	1553293	66239	17109
6	2004	35330	1695452	73670	17068
7	2005	35570	1709389	73652	16967
8	2006	35995	1818780	75326	16856
9	2007	37304	2017697	86697	17038
10	2008	38079	2087667	91299	17000
11	2009	38126	1943386	86239	17310
12	2010	38592	1997547	89457	17023
13	2011	38925	2048433	92903	16927
14	2012	38523	2133784	94902	16688
15	2013	38475	2135706	92001	16244

In [99]:

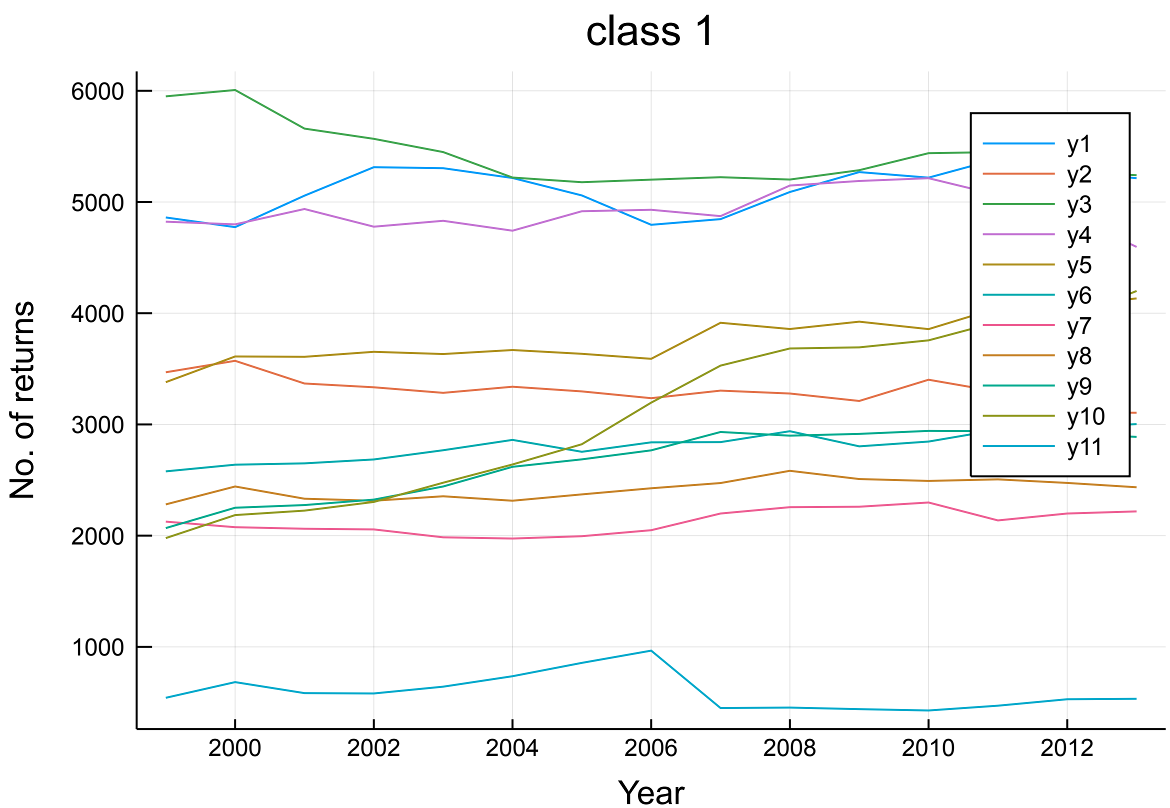
```
using Plots
x = tax_Tompkins_aggr_sum[:Year];
y = tax_Tompkins_aggr_sum[:ReturnCount_sum]; # These are the plotting data
plot(x,y, title="Year wise total number of returns",xlabel = "Year", ylabel = "No. of returns", leg=fals
```

Out[99]:



In [20]:
```
## repeat it for i equal 1 to 11 to get for all class.
    tax_Tompkins_income_class_1 =tax_Tompkins[tax_Tompkins[:Income_Class_Sort_Order] .== 11,:]
    #tax_Tompkins_income_class_1
    y = tax_Tompkins_income_class_1[:ReturnCount]
    x = tax_Tompkins_aggr_sum[:Year]
    plot!(x,y, title="class 1",xlabel = "Year", ylabel = "No. of returns", leg = true)
#tax_Tompkins_income_class_aggr = groupby(tax_Tompkins, :Income_Class_Sort_Order; sort = false, skipmis
```

Out[20]:

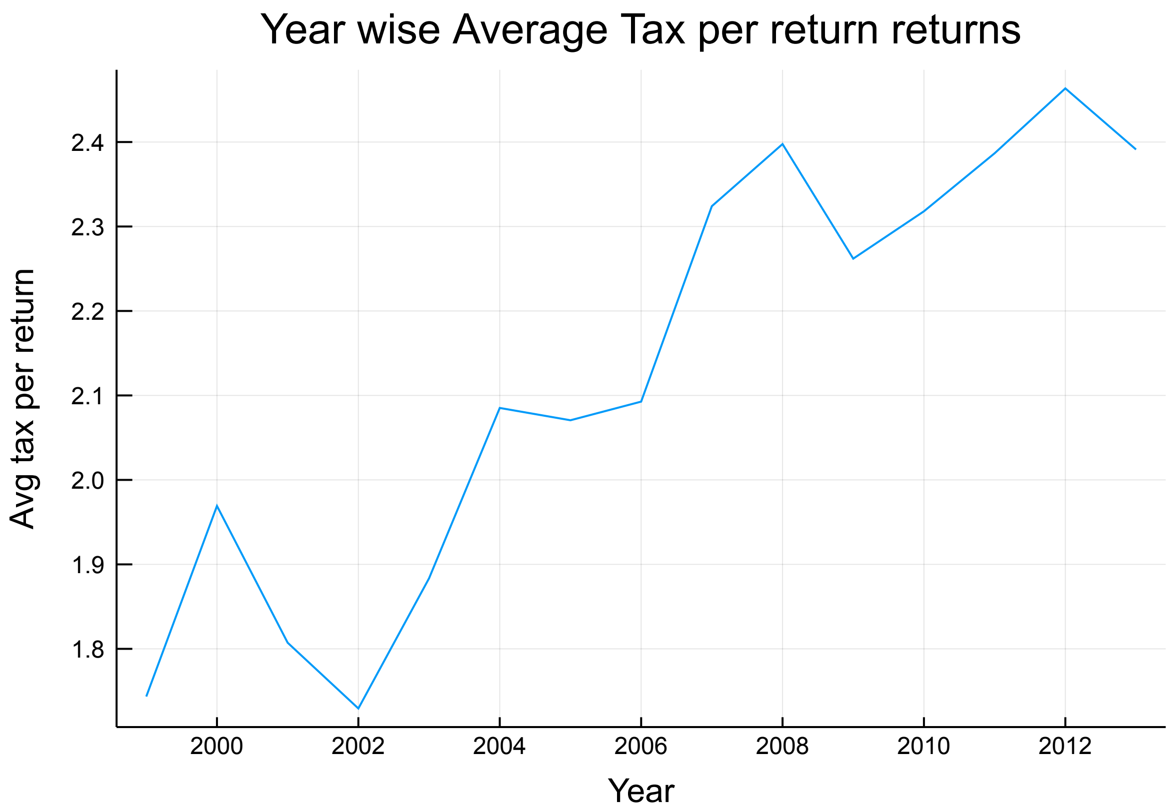


Please note that income class 12 and 13 has data only for 7 years (not all 15 years)

In [100]:
```
tax_Tompkins_aggr_sum[:tax_per_return] = tax_Tompkins_aggr_sum[:TotalTaxLiability_sum]./tax_Tompkins_ag
```

In [101]:
```
using Plots
x = tax_Tompkins_aggr_sum[:Year];
y = tax_Tompkins_aggr_sum[:tax_per_return]; # These are the plotting data
plot(x,y, title="Year wise Average Tax per return returns",xlabel = "Year", ylabel = "Avg tax per returi
```

Out[101]:



What kind of plot did you choose to make? Why?

I have used line graph to see if there is any year on year trend in these entities. Scatter or bar graph could also be used for this purpose.

answer b

Continuing to look only at Tompkins County, fit a model that predicts avg_tax using the year number. Do this with least squares.

In [102]:
```
function plotdata(x=x,y=y; margin=.05)
    scatter(x,y, label="data")
    xlabel!("x")
    ylabel!("y")
    range_y = maximum(y) - minimum(y)
    range_x = maximum(x) - minimum(x)
    ylims!((minimum(y)-margin*range_y,maximum(y)+margin*range_y))
    xlims!((minimum(x)-margin*range_x,maximum(x)+margin*range_x))
end
```

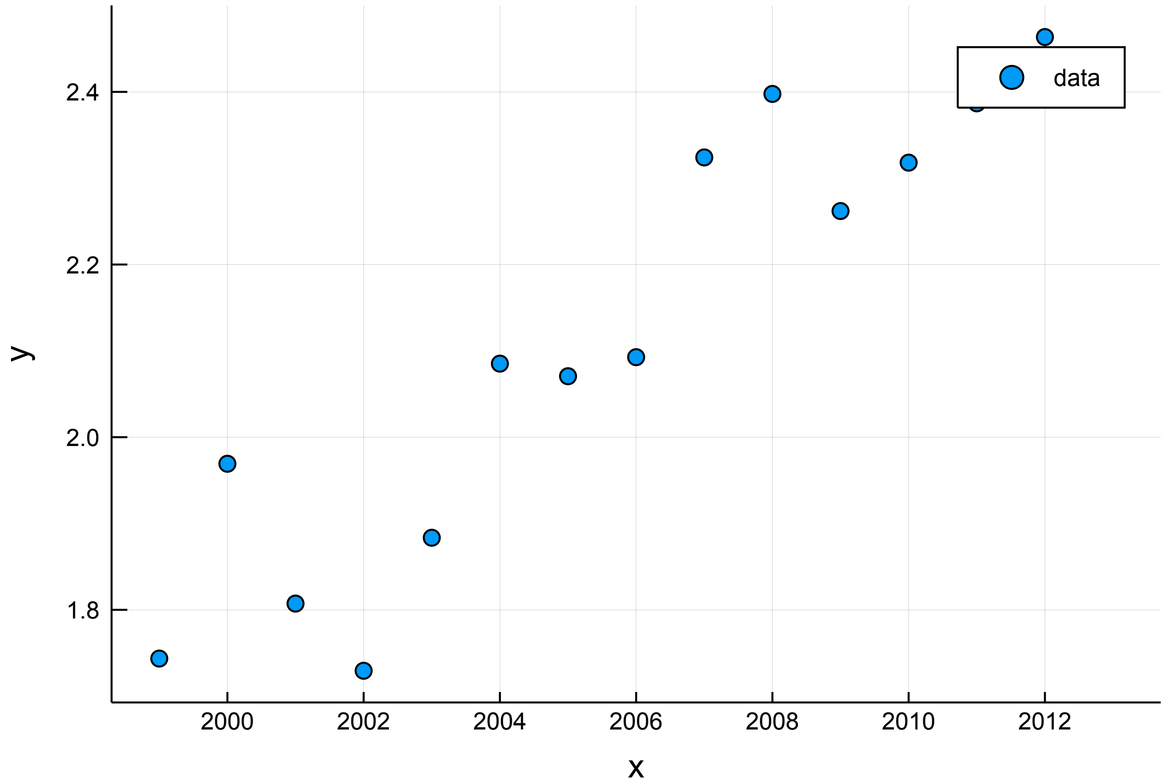
Out[102]:
```
plotdata (generic function with 3 methods)
```

In [103]:
```
y
```

Out[103]:
```
15-element Array{Float64,1}:
 1.7435400516795865
 1.9692048632912837
 1.8072386213245872
 1.7294107536738377
 1.8835020473157416
 2.0851967166713843
 2.0706213100927746
 2.0926795388248367
 2.32406712416899
 2.397620735838651
 2.261947227613702
 2.318019278606965
 2.386718047527296
 2.4635153025465306
 2.3911890838206626
```

In [104]:
```
plotdata(x,y)
```

Out[104]:



In [105]:
```
"""plot line y = w*x+b"""
function plotline(w,b;
                  xmin=-100,xmax=100,label="")
    xsamples = [xmin, xmax]
    plot!(xsamples, [w*x+b for x in xsamples], color=:black, label=label)
end
```

Out[105]:
```
plotline
```

In [106]:
```
X = [copy(x) ones(length(x))]
wb = X\y
```

Out[106]:
```
2-element Array{Float64,1}:
    0.052731666016322135
 -103.65142398187575
```

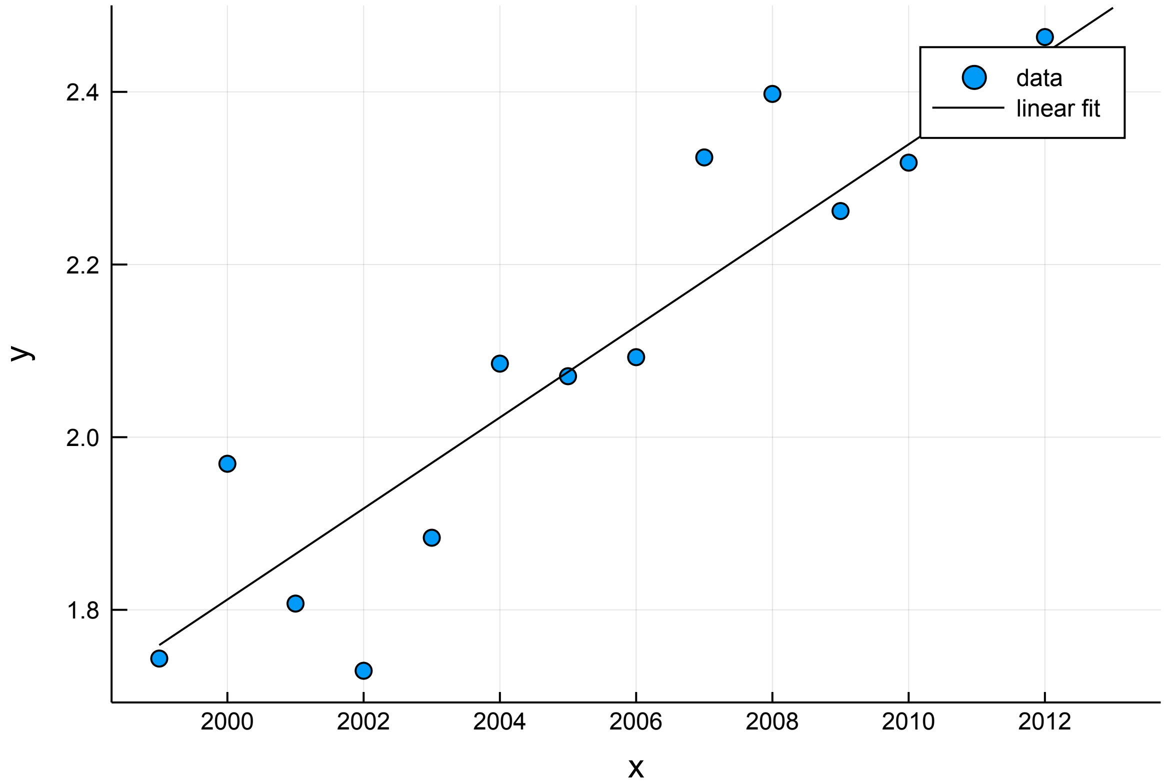
as seen above cofficient of year is 0.0527 and offset is -103.65

In [107]:
```
X
```

Out[107]:
```
15×2 Array{Union{Missing, Float64},2}:
 1999.0  1.0
 2000.0  1.0
 2001.0  1.0
 2002.0  1.0
 2003.0  1.0
 2004.0  1.0
 2005.0  1.0
 2006.0  1.0
 2007.0  1.0
 2008.0  1.0
 2009.0  1.0
 2010.0  1.0
 2011.0  1.0
 2012.0  1.0
 2013.0  1.0
```

In [108]:
```
plotline(wb[1], wb[2], xmin= minimum(X, dims=1)[1] ,xmax= xmin= maximum(X, dims=1)[1], label="linear fit
```

Out[108]:

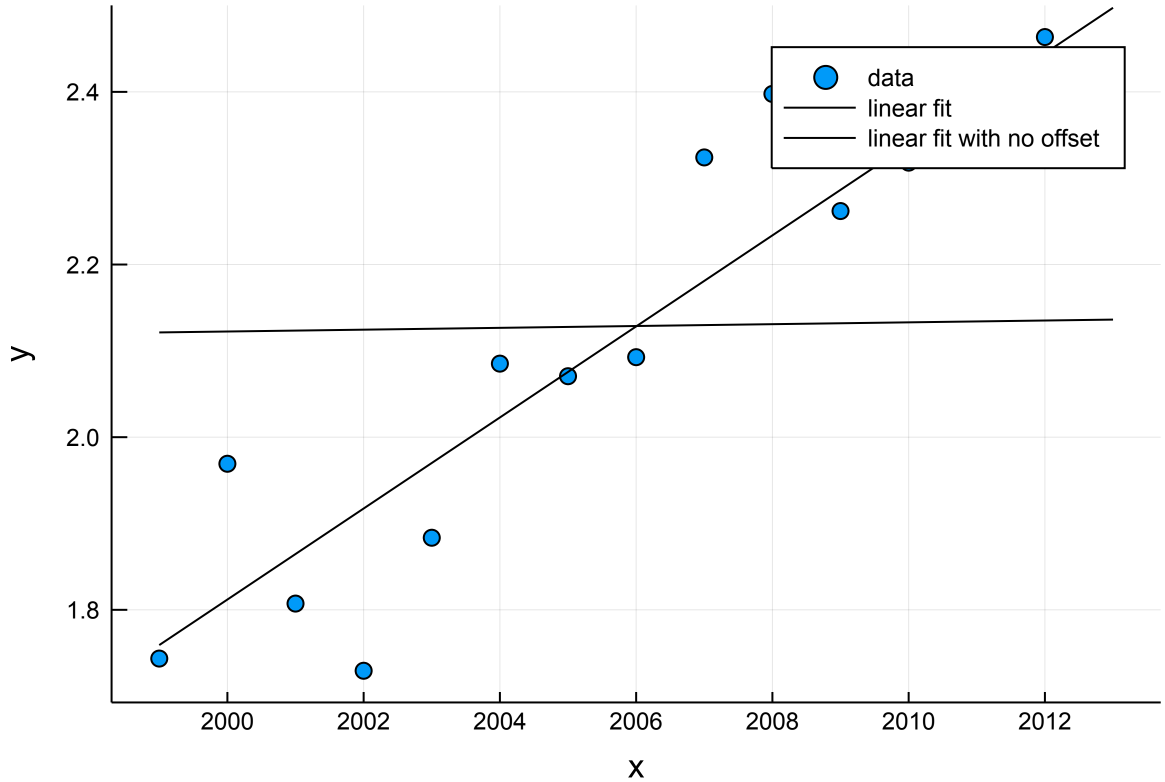


In [109]:
```
w = x\y
w
```

Out[109]:
```
0.0010612058135839807
```

In [110]:
```
plotline(w, 0, xmin= minimum(X, dims=1)[1] ,xmax= xmin= maximum(X, dims=1)[1], label="linear fit with no
```

Out[110]:



Answer c

In [111]:
```
X_part_c = tax_Tompkins_aggr_sum[:, [:Year, :tax_per_return]]
#print(X_part_c)

X_c = X_part_c[1:end,:]

for i in range(2,size(X_part_c)[1])
    X_c[i,1] = X_part_c[i,1]
    X_c[i,2] = X_part_c[i-1,2]
end
X_c_2 = X_c[2:end,:]

#print(X_c_2)

X_part_c_3 = [copy(X_c_2) ones(size(X_c_2)[1])]
X_part_c_final = convert(Matrix, X_part_c_3[:,1:3])
#print(X_part_c_final)
y_part_c_final = y[2:end]

#X_c = X[2:end,:]
#size(X_one_year_leg)
#y_one_year_leg = y[2:end]
```

Out[111]:
```
14-element Array{Float64,1}:
 1.9692048632912837
 1.8072386213245872
 1.7294107536738377
 1.8835020473157416
 2.0851967166713843
 2.0706213100927746
 2.0926795388248367
 2.32406712416899
 2.397620735838651
 2.261947227613702
 2.318019278606965
 2.386718047527296
 2.4635153025465306
 2.3911890838206626
```

In [112]:
```
X_part_c_final
```

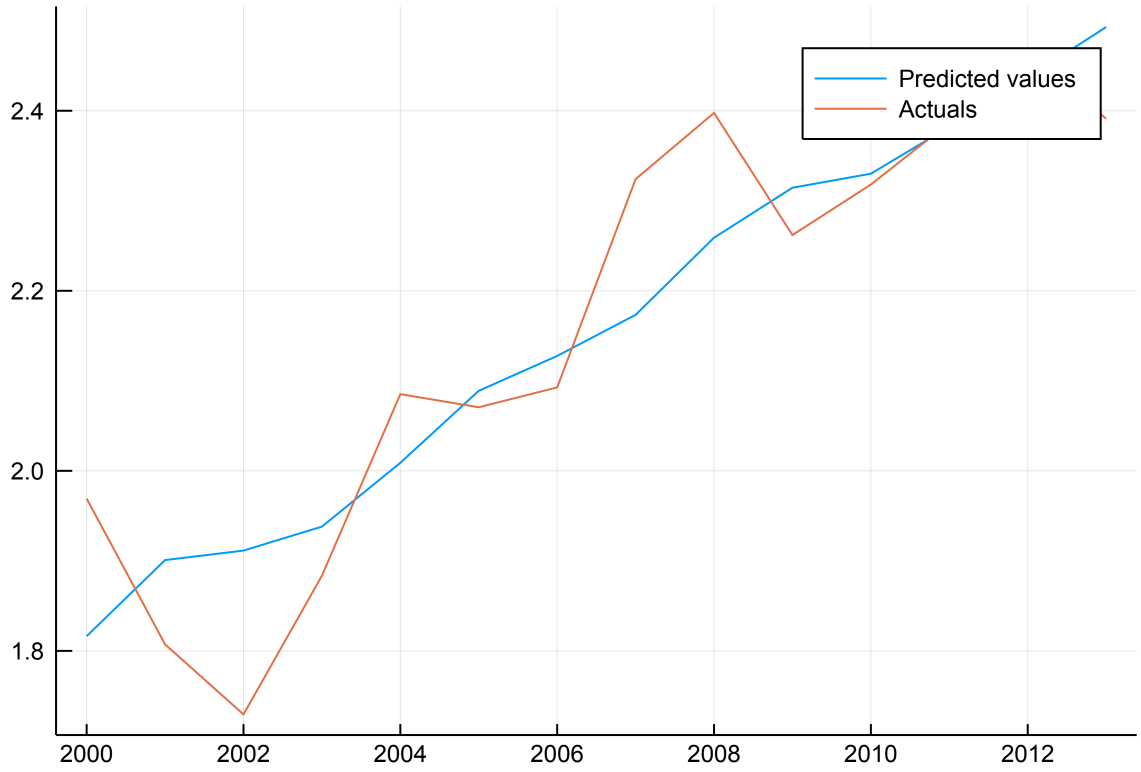
Out[112]:
```
14×3 Array{Union{Missing, Float64},2}:
 2000.0  1.74354  1.0
 2001.0  1.9692   1.0
 2002.0  1.80724  1.0
 2003.0  1.72941  1.0
 2004.0  1.8835   1.0
 2005.0  2.0852   1.0
 2006.0  2.07062  1.0
 2007.0  2.09268  1.0
 2008.0  2.32407  1.0
 2009.0  2.39762  1.0
 2010.0  2.26195  1.0
 2011.0  2.31802  1.0
 2012.0  2.38672  1.0
 2013.0  2.46352  1.0
```

In [113]:
```
wc = X_part_c_final \ y_part_c_final
```

Out[113]:
```
3-element Array{Float64,1}:
   0.04147719380205998
   0.19097044071089236
 -81.4710645892002
```

In [114]:
```
y_pred = X_part_c_final*wc
y_pred
plot(X_part_c_final[:,1], y_pred, label = "Predicted values")
plot!(X_part_c_final[:,1], y_part_c_final, label = "Actuals")
```

Out[114]:



As seen from above graph, model overpredicts in and around 2002 and underpredicts in around 2008. Prima facie, model does not model extreme values. it kinds of "averages out" to give a lot more smooth predicted curve - which seems to be underfitting actual values. Adding more features might help.

In [115]:
```
## Lets calculate average error on tompkins data:
avg_error_tompkins = mean((y_pred-y_part_c_final).^2)
avg_diff_tompkins = mean((y_pred-y_part_c_final))
print("average error of model c on tompkins data is ",avg_error_tompkins)
println()
print("average difference between y actual and y predicted is ",avg_diff_tompkins)
```

```
average error of model c on tompkins data is 0.009397861644978165
average difference between y actual and y predicted is -2.220446049250313e-16
```

Answer D

I will be adding total income of previous year and total exemptions of previous year as 2 additional features.

Total income in previous year can give some information about total taxable income this year, and hence tax being paid.

and exemption levels previous year determines expected exemption this year, which in turn gives some informaiton on loss in tax collected.

In [116]:

```
X_part_d = tax_Tompkins_aggr_sum[:, [:Year,:tax_per_return,:TotalIncome_sum, :TotalExemptions_sum]]
#print(X_part_c)

X_d = X_part_d[1:end,:]

for i in range(2,size(X_part_d)[1])
    X_d[i,1] = X_part_d[i,1]
    X_d[i,2] = X_part_d[i-1,2]
    X_d[i,3] = X_part_d[i-1,3]
    X_d[i,4] = X_part_d[i-1,4]
end
X_d_2 = X_d[2:end,:]

#print(X_c_2)

X_part_d_3 = [copy(X_d_2) ones(size(X_d_2)[1])]
X_part_d_final = convert(Matrix, X_part_d_3[:,1:5])
#print(X_part_c_final)
#y_part_c_final = y[2:end]
```

Out[116]:

```
14×5 Array{Union{Missing, Float64},2}:
 2000.0  1.74354  1.42013e6  17073.0  1.0
 2001.0  1.9692   1.58553e6  17193.0  1.0
 2002.0  1.80724  1.49376e6  16956.0  1.0
 2003.0  1.72941  1.47615e6  16894.0  1.0
 2004.0  1.8835   1.55329e6  17109.0  1.0
 2005.0  2.0852   1.69545e6  17068.0  1.0
 2006.0  2.07062  1.70939e6  16967.0  1.0
 2007.0  2.09268  1.81878e6  16856.0  1.0
 2008.0  2.32407  2.0177e6   17038.0  1.0
 2009.0  2.39762  2.08767e6  17000.0  1.0
 2010.0  2.26195  1.94339e6  17310.0  1.0
 2011.0  2.31802  1.99755e6  17023.0  1.0
 2012.0  2.38672  2.04843e6  16927.0  1.0
 2013.0  2.46352  2.13378e6  16688.0  1.0
```

In [117]:
```
wd = X_part_d_final \ y_part_c_final
wd
```

Out[117]:
```
5-element Array{Float64,1}:
   0.03441551039157461
  -0.7197304700729736
   1.0543875784963154e-6
   0.00015835947834075224
 -69.95532323364594
```

cofficient of year is about 0.0344.

cofficient of last year is about -0.7197

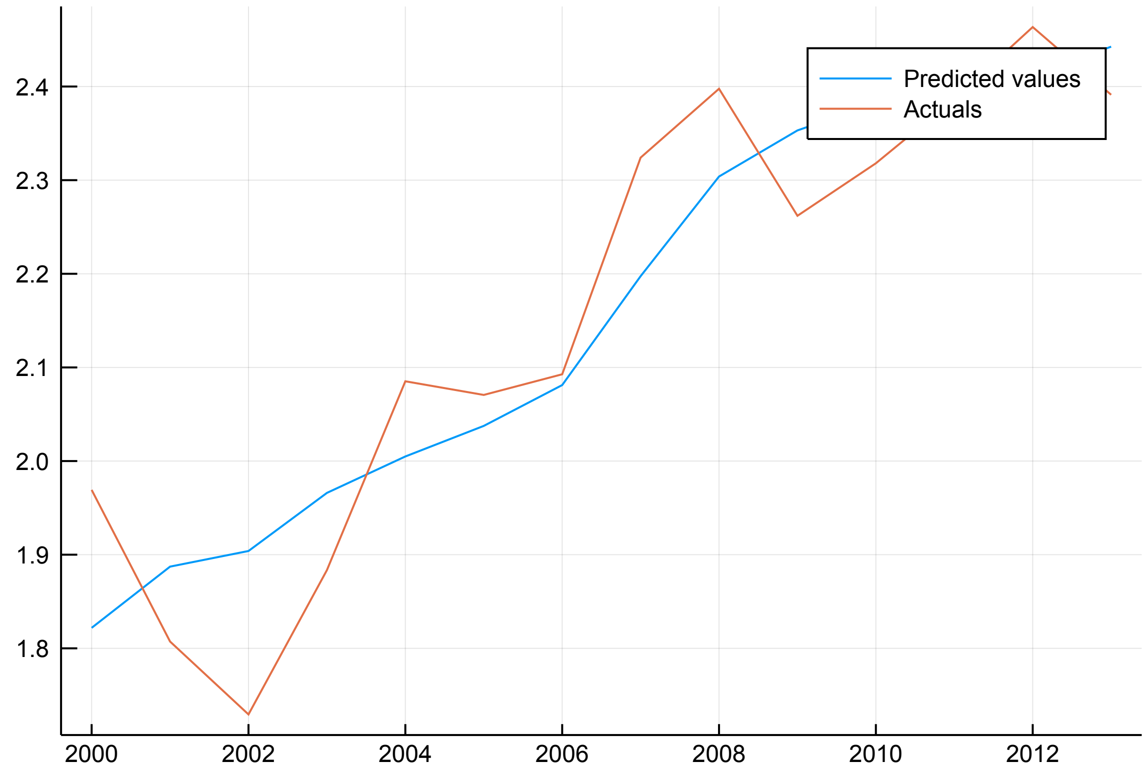
cofficient of last year's income is about 1.05*10-6 (scale of income is of order 10^6 as comapred to last year tax rate.)

cofficient of last year's exception is about 0.000158

offset term is -69.955.

In [118]:
```
y_pred_d = X_part_d_final*wd
y_pred_d
plot(X_part_c_final[:,1], y_pred_d, label = "Predicted values")
plot!(X_part_c_final[:,1], y_part_c_final, label = "Actuals")
```

Out[118]:



In [119]:
```
## Lets calculate average error on tompkins data:
avg_error_tompkins_d = mean((y_pred_d-y_part_c_final).^2)
avg_diff_tompkins_d = mean((y_pred_d-y_part_c_final))
print("average error of model d on tompkins data is ",avg_error_tompkins_d)
println()
print("average difference between y actual and y predicted is ",avg_diff_tompkins_d)
```

```
average error of model d on tompkins data is 0.008262071807l7244
average difference between y actual and y predicted is 2.6169542723307262e-14
```

Answer E

Yes, cofficient of year reduced slightly from 0.0414 to 0.0344. Which indicates that importance of year reduced when we added more features.

Answer F

First checking for counties that have data for all the years from 1999-2013

In [120]:
```
tax_minus_total_row = tax[tax[:IncomeClass].!="Total",:]
sort!(tax_minus_total_row, [order(:Year), order(:TotalTaxableIncome)])
tax_aggr_sum = by(tax_minus_total_row, [:Year, :County], :ReturnCount => sum, :TotalIncome => sum, :Tota
```

Out[120]: 974 rows × 6 columns

	Year	County	ReturnCount_sum	TotalIncome_sum	TotalTaxLiability_sum	TotalExemptions_sum
	Int64⍰	String⍰	Int64	Int64	Int64	Int64
1	1999	Hamilton	2534	73356	2505	1037
2	1999	NYS Unclassified +	3547	125119	4946	1486
3	1999	Residence Unknown ++	13056	1235334	63395	4046
4	1999	Schuyler	7571	228317	7470	4295
5	1999	Yates	9812	301294	9718	6088
6	1999	Lewis	10489	300015	8559	6936
7	1999	Schoharie	12454	392216	13202	6955
8	1999	Essex	15855	507403	17846	8100
9	1999	Seneca	13762	427067	14316	7944
10	1999	Greene	18869	628309	22938	9726
11	1999	Orleans	17064	524691	17326	10748
12	1999	Delaware	19195	570872	18862	9901
13	1999	Franklin	18079	514622	15828	10882
14	1999	Cortland	19114	601623	20176	11094
15	1999	Allegany	17465	503187	15416	10935
16	1999	Montgomery	21622	643378	21232	11559
17	1999	Fulton	22949	704502	23902	12437
18	1999	Chenango	20634	651087	21773	12307
19	1999	Wyoming	17223	575074	19785	10532
20	1999	Tioga	21435	749034	26143	13287
21	1999	Washington	24206	742371	23171	13849
22	1999	Otsego	24127	808971	29363	12932

	Year	County	ReturnCount_sum	TotalIncome_sum	TotalTaxLiability_sum	TotalExemptions_sum
	Int64⍰	String⍰	Int64	Int64	Int64	Int64
23	1999	Columbia	26382	1051654	40551	13833
24	1999	Sullivan	28801	987096	35553	15928
25	1999	Livingston	26129	931228	34632	15461
26	1999	Madison	28024	1015613	37928	17174
27	1999	Herkimer	26706	769293	24099	14992
28	1999	Clinton	31044	1044671	37560	17571
29	1999	Genesee	26686	879585	31089	15809
30	1999	Warren	28438	1046895	41320	14791
⋮	⋮	⋮	⋮	⋮	⋮	⋮

In [121]:

```
counties = Set((tax[:County]))
data_available_counties = String[]
data_messy_counties = String[]

for county in counties
    # println("$county reported for $(length(crime[crime[:County].==county,:County])) years.")
    if length(tax_aggr_sum[tax_aggr_sum[:County].==county,:County]) == 15
        print(county, "1")
        push!(data_available_counties, county)
    else
        push!(data_messy_counties, county)
    end
end
data_available_counties
```

```
Cattaraugus1Onondaga1Otsego1Orleans1Niagara1Cortland1Westchester1Herkimer1Clinton1Rockland1Sullivan1Wa
shington1Schoharie1Allegany1New York City - Queens1Franklin1Saratoga1Jefferson1Nassau1Rensselaer1Madis
on1Essex1Ontario1Erie1Ulster1Broome1Wayne1Lewis1Schenectady1Tompkins1Chemung1Columbia1Oneida1Wyoming1C
ayuga1New York City - Bronx1Seneca1Dutchess1Chenango1Schuyler1Oswego1Albany1New York City - Kings1Dela
ware1Hamilton1Montgomery1Yates1Chautauqua1New York City - Richmond1Livingston1Residence Unknown ++1NYS
Unclassified +1Warren1Greene1Suffolk1Steuben1Putnam1Monroe1Orange1Fulton1Genesee1Tioga1New York City -
Manhattan1St. Lawrence1
```

Out[121]:

```
64-element Array{String,1}:
 "Cattaraugus"
 "Onondaga"
 "Otsego"
 "Orleans"
 "Niagara"
 "Cortland"
 "Westchester"
 "Herkimer"
 "Clinton"
 "Rockland"
 "Sullivan"
 "Washington"
 "Schoharie"
 ⋮
 "Warren"
 "Greene"
 "Suffolk"
 "Steuben"
 "Putnam"
```

```
"Monroe"
"Orange"
"Fulton"
"Genesee"
"Tioga"
"New York City - Manhattan"
"St. Lawrence"
```

As can be seen above we have 64 counties having full data in this period.

In [122]:
```
### Looping process in part C for these 64 counties.

avg_error_wc = zeros(length(data_available_counties))
for i in range(1, length(data_available_counties))
    data = tax_aggr_sum[tax_aggr_sum[:County].== data_available_counties[i],:]
    data[:tax_per_return] = data[:TotalTaxLiability_sum]./data[:ReturnCount_sum];
    avg_tax_col = data[:tax_per_return]
    y_actual = avg_tax_col[2:end]



    X_part_c = data[:, [:Year, :tax_per_return]]
    X_c = X_part_c[1:end,:]
    for i in range(2,size(X_part_c)[1])
        X_c[i,1] = X_part_c[i,1]
        X_c[i,2] = X_part_c[i-1,2]
    end
    X_c_2 = X_c[2:end,:]

    #print(X_c_2)
    X_part_c_3 = [copy(X_c_2) ones(size(X_c_2)[1])]  ## adding offset
    X_part_c_final = convert(Matrix, X_part_c_3[:,1:3]) ## converting to matrix

    y_pred = X_part_c_final*wc
    #print(X_part_c_final)

    avg_error_wc[i] = mean((y_pred-y_actual).^2)
end
print(avg_error_wc)
```

```
[0.795158, 0.0446452, 0.392618, 0.806688, 0.360154, 0.545696, 14.4433, 0.763728, 0.400323, 0.394645,
0.364568, 0.721901, 0.523369, 0.830908, 0.430079, 0.772131, 0.102861, 0.595961, 3.83213, 0.142621, 0.2
11277, 0.441726, 0.0441445, 0.0665735, 0.0747806, 0.339714, 0.398547, 0.879635, 0.0848177, 0.00939786,
0.328549, 0.0455119, 0.426888, 0.614655, 0.4951, 1.38451, 0.64944, 0.10737, 0.792012, 0.730738, 0.4985
02, 0.0111474, 0.324029, 0.687634, 0.824336, 0.731485, 0.813826, 0.823021, 0.0116939, 0.359979, 156.42
6, 1.07707, 0.120057, 0.241102, 0.778848, 0.324168, 1.04328, 0.0274688, 0.00734097, 0.666572, 0.47122
9, 0.395658, 38.3612, 0.623563]
```

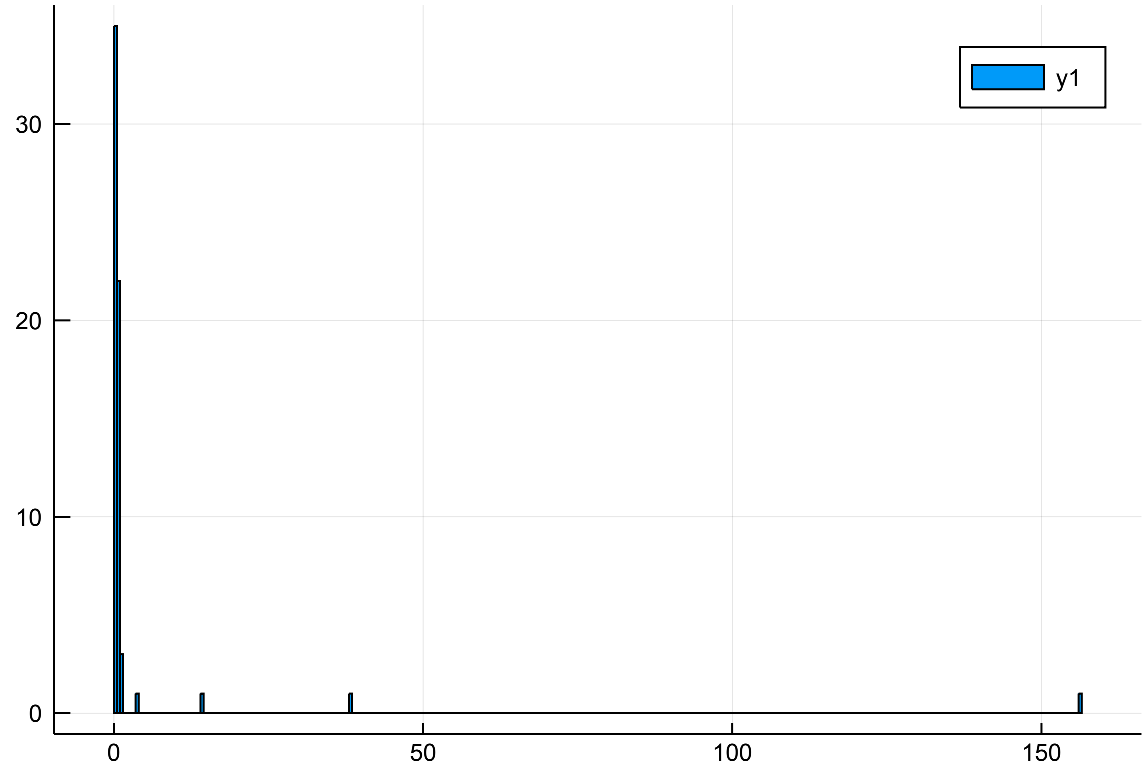
In [123]:
```
using PyPlot
```

In [124]:
```
maximum(avg_error_wc)
```

Out[124]: 156.42566528178685

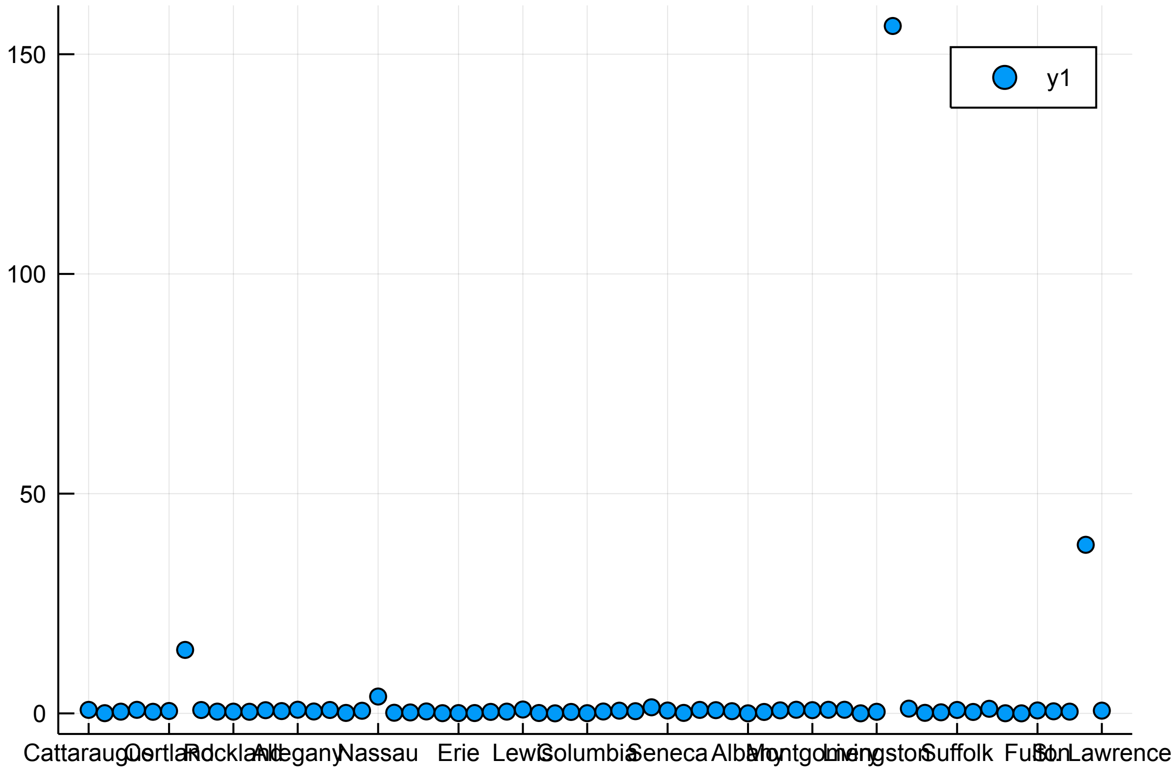
In [125]:
```
histogram(avg_error_wc)
```

Out[125]:



In [126]:
```
scatter(data_available_counties,avg_error_wc)
```

Out[126]:



In [127]:
```
data_available_counties[avg_error_wc.> 1]
```

Out[127]:
```
7-element Array{String,1}:
 "Westchester"
 "Nassau"
 "New York City - Bronx"
 "Residence Unknown ++"
 "NYS Unclassified +"
 "Putnam"
 "New York City - Manhattan"
```

In [128]:
```
data_available_counties[avg_error_wc.< 0.009]
```

Out[128]:
```
1-element Array{String,1}:
 "Orange"
```

As seen from graph above, Counties: {"Westchester", "Nassau", "New York City - Bronx", Putnam", "New York City - Manhattan"} are outlier for this model.

average error of model c on tompkins data is 0.009397861644978165. - About 61 counties have more error than this. Only 1 country (Orange) has less error than this.

Answer G

In [129]:

```
avg_error_G_wd = zeros(length(data_available_counties))
G_wd_array = zeros(length(data_available_counties),4)
for i in range(1, length(data_available_counties))
    data_g = tax_aggr_sum[tax_aggr_sum[:County].== data_available_counties[i],:]
    data_g[:tax_per_return] = data_g[:TotalTaxLiability_sum]./data_g[:ReturnCount_sum];
    avg_tax_col_g = data_g[:tax_per_return]
    y_actual_g = avg_tax_col_g[2:end]

    X_part_d_g = data_g[:, [:Year,:tax_per_return,:TotalIncome_sum, :TotalExemptions_sum]]
    X_d_g = X_part_d_g[1:end,:]
    for i in range(2,size(X_part_d_g)[1])
        X_d_g[i,1] = X_part_d_g[i,1]
        X_d_g[i,2] = X_part_d_g[i-1,2]
        X_d_g[i,3] = X_part_d_g[i-1,3]
        X_d_g[i,4] = X_part_d_g[i-1,4]
    end
    X_d_2_g = X_d_g[2:end,:]

    #print(X_c_2)
    X_part_d_3_g = [copy(X_d_2_g) ones(size(X_d_2_g)[1])]  ## adding offset
    X_part_d_final_g = convert(Matrix, X_part_d_3_g[:,1:3]) ## converting to matrix

    wd_G = X_part_d_final_g \ y_actual_g

    y_pred_g = X_part_d_final_g*wd_G
    #print(X_part_c_final)

    avg_error_G_wd[i] = mean((y_pred_g-y_actual_g).^2)
    print(wd_G)
    println()

end
print(avg_error_G_wd)
```

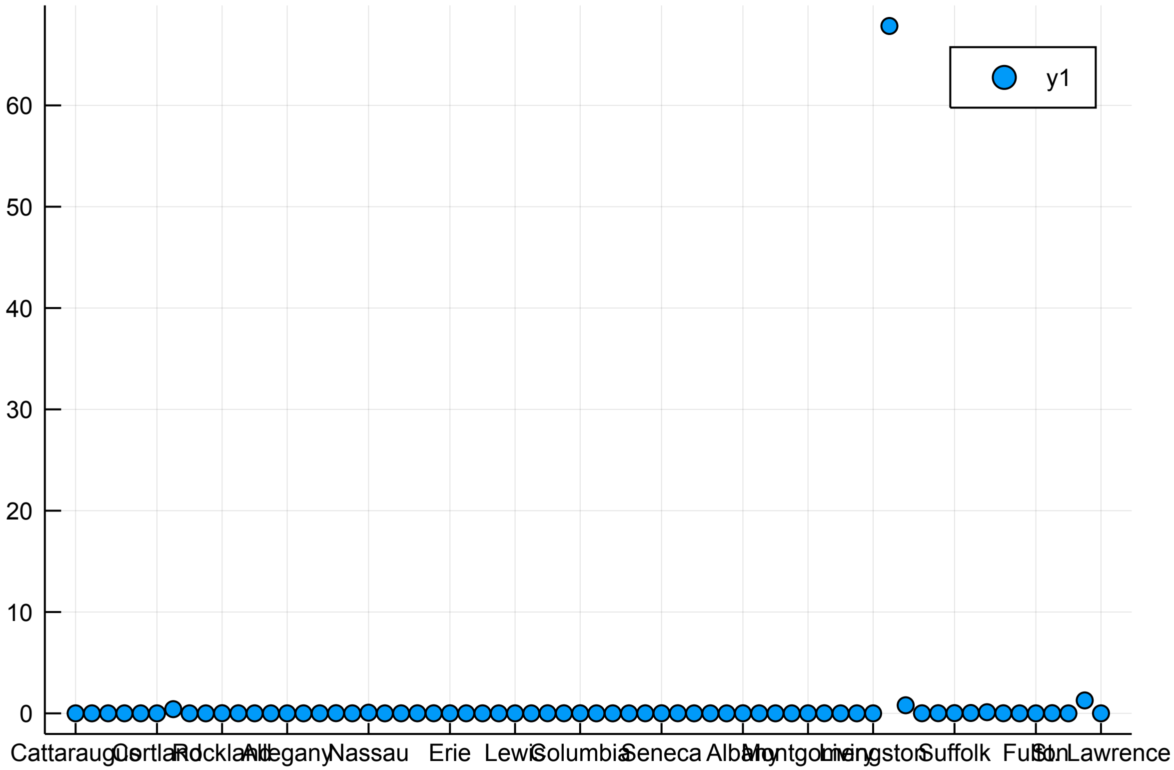
```
[0.000105664, 0.322759, 4.63283e-7]
[0.000107817, 0.664388, 4.69291e-8]
[5.28468e-5, 0.744305, 2.86605e-7]
[0.000369276, 0.205306, 1.91051e-7]
[5.31499e-5, 0.762242, 6.81192e-8]
[0.000105767, 0.368179, 8.19548e-7]
[0.0013662, 1.50618, -1.15865e-7]
[-2.4971e-5, 0.308668, 9.02866e-7]
[0.000124519, 0.383779, 4.7607e-7]
```

```
[0.000456102, 0.629347, 2.0978e-8]
[0.000224725, 0.389649, 3.48194e-7]
[0.000317177, -0.894187, 1.6426e-6]
[6.29733e-5, 0.759752, 4.35837e-7]
[5.26449e-5, -0.0550298, 1.65559e-6]
[0.000388302, -0.153452, 2.2086e-8]
[7.73586e-5, 0.167006, 1.19929e-6]
[0.00018807, 0.309495, 2.60484e-7]
[0.000111398, -0.0625421, 7.64124e-7]
[0.000758195, 1.40551, -5.91645e-8]
[0.000194629, 0.229641, 3.08752e-7]
[-6.35215e-6, 0.439877, 7.71784e-7]
[0.000125893, 1.06336, -4.98973e-7]
[7.91793e-5, 0.630559, 2.6609e-7]
[9.76559e-5, 1.05095, -1.37842e-8]
[0.000125594, 0.898396, -9.7058e-9]
[3.0549e-5, 0.622049, 1.46749e-7]
[0.000304707, 0.227576, 2.89346e-7]
[2.76709e-5, 0.702218, 7.36063e-7]
[0.000269805, 0.357894, 2.0504e-7]
[0.000445666, -0.74301, 1.58543e-6]
[0.000172848, 0.453005, 3.16467e-7]
[0.000250842, 1.43055, -9.92297e-7]
[9.92256e-5, 0.328078, 1.94465e-7]
[4.71868e-5, -0.0624562, 1.87179e-6]
[3.29843e-5, 0.940851, 3.00248e-8]
[-0.000252011, 1.28533, 2.17075e-8]
[0.000170154, 0.052991, 1.53377e-6]
[0.000316622, 0.613994, 5.31238e-8]
[0.000113249, 0.284764, 7.56026e-7]
[-1.04907e-5, 0.561379, 1.93119e-6]
[7.23066e-5, 0.0140817, 6.14232e-7]
[0.000167848, 0.982708, -3.71138e-8]
[1.69049e-5, 0.677074, 1.21895e-8]
[3.93097e-5, 0.264031, 1.18778e-6]
[0.000211273, 0.944, -4.57284e-6]
[0.000232303, 0.0736301, 7.34227e-7]
[0.000196267, -0.0357345, 2.0246e-6]
[0.000154206, -0.0317764, 4.1618e-7]
[0.000398724, 0.418453, 4.88829e-8]
[8.15341e-5, 0.520394, 5.09143e-7]
[0.00182191, 0.0885821, 4.08015e-6]
[0.00139693, -0.387855, 1.10357e-6]
```

```
[0.000183576, 0.871846, -8.89194e-8]
[0.000106238, 0.350201, 1.02652e-6]
[0.000248708, 1.08224, -1.53874e-8]
[0.000599216, 0.136589, 6.72924e-8]
[0.000825052, -0.66462, 1.21762e-6]
[0.000228902, 0.768982, 1.22904e-9]
[0.000271008, 0.593879, 4.10545e-8]
[0.000396796, -0.274974, 8.50817e-7]
[-7.50782e-5, -0.432146, 2.05231e-6]
[0.000180111, -0.318972, 1.6738e-6]
[0.0010465, 1.1803, -2.85515e-8]
[2.84961e-5, 0.442147, 4.34634e-7]
[0.00214576, 0.00323026, 0.004329, 0.0037459, 0.00265191, 0.00156007, 0.413695, 0.00425857, 0.0011863
4, 0.0149486, 0.00986826, 0.00402088, 0.00271592, 0.00118995, 0.00170081, 0.00475299, 0.0174434, 0.003
67631, 0.0686952, 0.00225045, 0.00345023, 0.00725515, 0.00783985, 0.00657344, 0.00540202, 0.00270273,
0.00192676, 0.00349628, 0.00287172, 0.0102424, 0.00763981, 0.00923982, 0.00235417, 0.00198876, 0.00187
246, 0.00726205, 0.00943497, 0.00991133, 0.00191292, 0.00319093, 0.00111505, 0.00892808, 0.00534022,
0.00355179, 0.00265264, 0.00500858, 0.00986382, 0.0019241, 0.00272912, 0.00197464, 67.8367, 0.797739,
0.00842079, 0.015314, 0.0221807, 0.0368529, 0.113873, 0.00950344, 0.00259395, 0.00145685, 0.0195711,
0.00115246, 1.27785, 0.000658831]
```

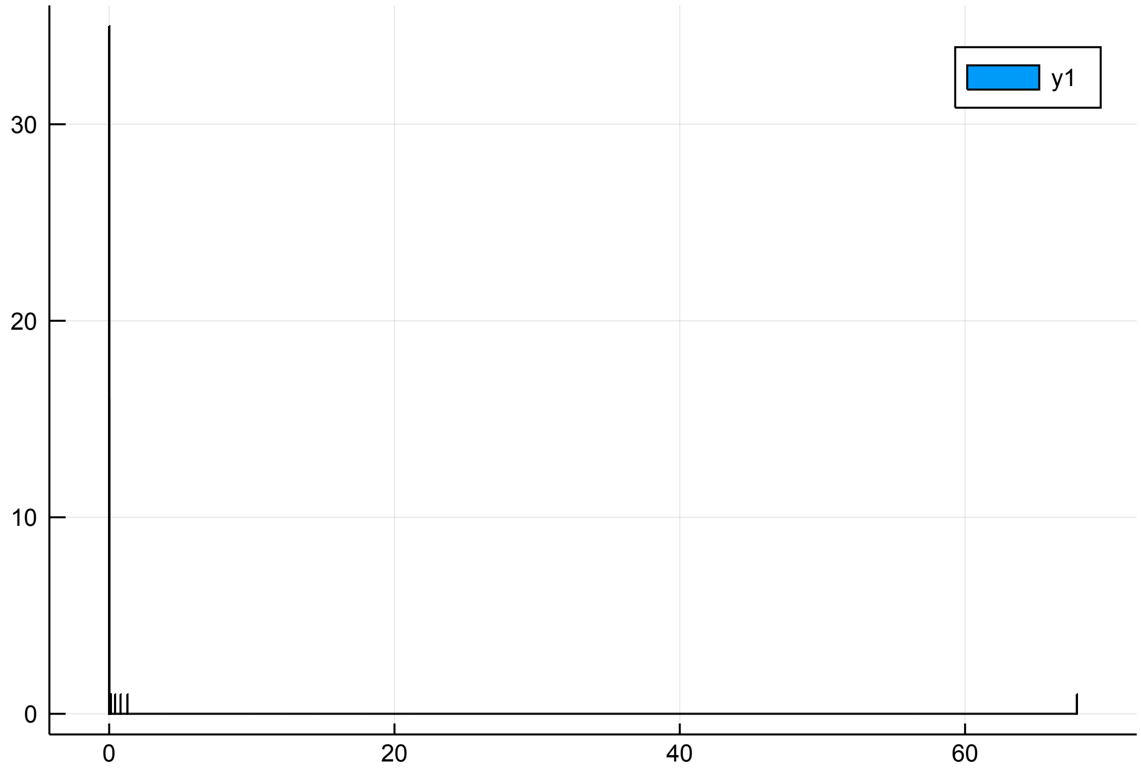
In [130]:
```
scatter(data_available_counties,avg_error_G_wd)
```

Out[130]:



In [131]:
```
histogram(avg_error_G_wd)
```

Out[131]:



How does this error distribution compare to that of the model you fit on Tompkins County?

This histogram is lot more concentrated on left hand side as compared to histogram in previous part. This indicates errors are smaller in magnitude and more consistent.

Are the coefficients of the model about the same for each county, or do they differ significantly?

Following are sample county specific wd for some of the counties:

[0.000105664, 0.322759, 4.63283e-7]

[0.000107817, 0.664388, 4.69291e-8]

[5.28468e-5, 0.744305, 2.86605e-7]

[0.000369276, 0.205306, 1.91051e-7]

[5.31499e-5, 0.762242, 6.81192e-8]

[0.000105767, 0.368179, 8.19548e-7]

[0.0013662, 1.50618, -1.15865e-7]

[-2.4971e-5, 0.308668, 9.02866e-7]

[0.000124519, 0.383779, 4.7607e-7]........

as seen above, county specific wd vectors varies significantly for each county.

Answer H

In [ ]:

```

```

In [132]:
```julia
data_available_counties[avg_error_G_wd.< 0.009]
```

```
 "Herkimer"
 "Clinton"
 "Washington"
 "Schoharie"
 "Allegany"
 "New York City - Queens"
 "Franklin"
 ⋮
 "New York City - Kings"
 "Delaware"
 "Hamilton"
 "Montgomery"
 "Chautauqua"
 "New York City - Richmond"
 "Livingston"
 "Warren"
 "Orange"
 "Fulton"
 "Tioga"
 "St. Lawrence"
```

Number of counties with error less than 0.009 has increased from previous 1 to 45.

Country specific models makes more sense.

If you wanted to predict the income tax in each county in future years, do you think the county-specific models or the Tompkins model would perform better? Why?

I will prefer county-specific models over Tompkins model as errors are on lower side and more consistent with county specific models. However, as discussed below, there is some down side of using county specific models.

What concerns might you have about each model?

1. Data required will be gathered at very slow rate (one data point in a year)

2. Maintenance of so many different models might be tricky and time.

3. Due to lack of data points there is always a chance of overfitting.

Answer I

What other information would you want to use to make your model even better?

I think if we have information about similiarity aspects of counties, that would solve for sparse data point issue. some unsupervised models can help understand if some counties are more similar to each other as compared to another.

Other lifestyle data like avg house hold income, hospital/other public resources per capita, local taxation laws, industry of population etc will also help to develop more detailed features.

Question 5

Calibration. How long did you spend on each problem in this homework assignment,and on the homework assignment, in total?

Question 1: 1 hour.

Question 2: 2 hour.

Question 3: 2 hours.

Question 4: 5 hour.

Total : 10 hours

In [ ]:

```
```

In [ ]:

```
```